FORECASTING ALGAE BLOOMS IN AQUACULTURE USING MUSSELS' OPENINGS DATA

by

Deepan Shankar Pondichery Vellamuthu Kripashanker

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2021

The thesis is dedicated to my family who have been the great support to me for this long journey.

Table of Contents

List of Tables

List of Figures

Abstract

Time series data consists of a series of measurements collected over a period of time. This type of data is very relevant in several domains, including healthcare, manufacturing, finance, environment, and many more. For these domains it is frequently of key importance to be able to predict the future values of these time series. Activity monitoring is a task related to forecasting where time series data is used as input signals to some events whose occurrence is supposed to depend on the values of these series. These events are typically of critical importance to the end users and the goal is to be able to anticipate them with sufficient lead time. Due to the uncertainty of the future, forecasting and anticipating these scenarios could help prevent or mitigate hazardous activities. In this thesis we address one such application - the anticipation of algae blooms in aquaculture industries. As algal blooms hinders the growth of the aquaculture species, it is of high importance to monitor the farms, avoiding serious damages to the species. In this thesis we propose a method for anticipating algae blooms based on measurements of mussels' valve openings that domain experts think can be used as bio-sentinels of the blooms. We use machine learning models to address this predictive task and obtain models that can predict future algal bloom events based on the micro closures of the mussels. We focus our goal on predicting the presence of the algae *Alexandrium Tamarense* in the water environment. Due to the rarity of algae blooms, sampling procedures were used to balance the distribution of the target variable to facilitate the task of the learning algorithms. Overall, the experimental comparisons we have carried out have shown that we were able to obtain very good results, particularly in terms of being able to anticipate a high percentage of the blooms (80%) although with some false alarms (48%). Our results have also shown the advantage of adding sampling procedures to overcome the imbalanced distribution of our target variable. In summary, in this thesis we have developed a series of forecasting approaches based on feature engineering, machine learning models and sampling methods that have shown a great potential in terms of preventing algae blooms in aquaculture farms.

Acknowledgements

I wish to thank my supervisor Dr. Luis Torgo who has been in support with my thesis. He has been a strong support and great sense of hope for my thesis. His guidance throughout my thesis helped me to solve critical problems and to shape my thesis. Also very thankful to other professors in the Data Analytics Lab for their support and guidance in my work. And to mention Vitor Cerqueira, post doctorate student working with Dr. Luis Torgo who has contributed as a mentor in my thesis. I wish to thank DeepSense in funding my thesis and providing me great support. I also wish to thank my family in all support.

Chapter 1

Introduction

In today's world, massive amounts of data are produced continuously. Many organizations are evolving to make use of this enormous amount of data that are produced. Organizations are finding ways to capture the data and analyze it to make data-driven decisions. In many cases, the data is generated as a time series, i.e., a set of values measured over a time period.

The main challenges in applications involving time series data is to accurately analyze the process that generates these types of data and make decisions accordingly. Many organizations try to incorporate the time series data and make decisions by predicting the future values of these time series. Forecasting denotes the process of estimating the future values (for example, y_{n+1}) from a time series denoted by $Y = \{y_1, y_2, ..., y_n\}$, where y_i represents the measured value of variable Y at time i. Forecasting the values of observations is one of the main objectives when modeling time series data. Forecasting is useful in many domains including healthcare, where for instance patient's blood pressure is monitored continuously to make decisions [38], finance [22, 40] and many others.

Many classical and machine learning methodologies have been developed to forecast and predict the future values of time series. However, it is widely accepted that no specific method is universally the best for all types of problems [23]. This is consistent with the well-known *No Free Lunch* theorem for supervised learning, which states exactly that there is no single algorithm that is most appropriate for all tasks [79].

In certain domains, we may wish to not only forecast the future values of the series, but also to anticipate specific events [20] that are associated with these values. This type of predictive task is usually known as activity monitoring [30]. It involves identifying certain events that might occur in the future in a proactive manner. Specifically, these events are identified when certain values of some time series cross pre-defined thresholds that may require action from domain professionals. Such early forecast of events is crucial in certain domains as they empower the organizations to take appropriate actions to prevent or mitigate the damages that these events may cause.

Activity monitoring is helpful in many domains like for instance health-care, where an event may occur if some health parameters of a patient cross some thresholds. For example, an acute hypotensive episode (AHE) event is triggered if 90% of the mean arterial blood pressure values are below 60 millimeters of mercury during a 30-minute interval being important to treat the patient to avoid causing organ damage, which may happen if not treated in a timely manner. [39]. Anticipating this type of events can also be helpful in wildlife photography, where the camera is triggered for movements of animals or birds by identifying them early, rather than recording it for a long time [51]. In activity monitoring, the main objective is not to accurately predict the future values of a time series but to anticipate events that are associated with these values and that may occur in the near future.

Frequently, the major challenge in activity monitoring tasks is that the events of interests are rare, and learning the task of identifying those events is regarded as an imbalanced learning problem and certain forecasting models find it hard to predict the rare events [17]. Another challenge in activity monitoring is anticipating the events with sufficient lead time, which benefits the organizations to assess the situation and take appropriate actions before the events occur.

In this thesis, we address a concrete activity monitoring task: to forecast or anticipate algae blooms in the context of aquaculture. Algae blooms are impactful events that seriously damage water quality and can have serious consequences like the dying of fish in the aquaculture farms. Domain experts experimented with placing certain mussels in the aquaculture farms with sensors attached to their valves. They concluded that the mussels behaved differently with the presence of algae in the water. More specifically, if these mussels show a behaviour known as micro closures this is considered to be an indicator of potential presence of algae in the water. Using data collected from sensors in mussels' valves, we analyze their behaviour in terms of micro closures, and try to use this information as predictors for the machine learning models that were able to forecast the algal blooms. We apply several machine learning models (SVM [27], Random Forest [18] and XGBoost [25]) and compare them on the task of detecting the algal bloom events based on the mussels' behaviour.

Due to the damaging effects of algal blooms in the health of fish, aquaculture farms need to avoid and/or quickly treat these blooms. In this context, forecasting algae blooms is extremely relevant for aquaculture farms as it may allow treating the water well in advance of the algal bloom occurrence. We have designed forecasting tasks to predict the future occurrence of algal blooms in aquaculture using information on micro closures in mussels' valve openings. If successful, these forecasting models have as main advantage the ability of avoiding the need to collect water samples and analyse them in the lab to check for the presence of algae. By using only information from the sensors at the mussels valves, our models allow aquaculture farms to save money and resources while still being able to anticipate and treat algal blooms.

We use feature engineering techniques to manipulate the original valve opening data with the goal of enriching the information provided to the models. We also use sampling methodologies to balance our dataset which primarily consists of normal events as opposed to the rare and more interesting algae bloom events. We try to forecast algal bloom events for future timestamps of $t+1$, $t+4$ and $t+8$ which correspond to 30 minutes, 2 hours and 4 hours.

We evaluate the proposed methods using metrics such as F-score, Precision and Recall, which are more useful on scenarios with imbalanced class distributions. We tune specific parameters of the different machine learning models and sampling methods to achieve better results on the above mentioned metrics. Finally, we compare the results among the different models and discuss the statistical significance of the results.

In summary, the main contributions of this thesis are,

- An analysis of the potential of using information on mussels closure behaviour as predictors to be used in prediction models for forecasting algae blooms in aquaculture farms
- An application of machine learning models to forecast future algal bloom events with different forecasting horizons
- An experimental comparison of different machine learning models in the task

of forecasting algae blooms

The thesis is organized as follows. Chapter 2 presents the background for the work of the thesis as well as related works. In Chapter 3 we introduce the case study that motivates the work of the thesis. The proposed methods for forecasting algae blooms are described in Chapter 4. In Chapter 5 we describe the experimental evaluation of our proposal, while the conclusions and future work are described in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, we present the background and related work to the topics addressed in this thesis. It is organized into the following sections. In Section 2.1 we start by defining the basic concepts of time series, its types and the problems. In Section 2.2, we introduce the concepts of forecasting time series, challenges and current state of the art approaches. This is followed by Section 2.3 which discusses forecasting model evaluation and estimation methods. In Section 2.4, we address the topic of activity monitoring, its challenges, related works and evaluation metrics. Finally, Section 2.5 summarizes the concepts of time series, forecasting and activity monitoring used in this thesis.

2.1 Time Series

A time series is defined as a set of values occurring sequentially during a period of time. For example, let Y be a time series of values, $Y = \{y_1, y_2, ..., y_n\}$ where each y_i is a value of Y measured at time i. For instance, the daily stock prices of a company can be seen as a time series. Time series can be regular, if the measurements are recorded at equally spaced points in time, or irregular, when the time interval between each measurement is not regular. Time series data occurs in many relevant application domains, including healthcare informatics where for instance we could have a time series of the blood pressure values of a patient (e.g. [37, 39]), stock market predictions (e.g. [22, 40]), energy consumption (e.g. [9]), intelligent transportation systems (e.g. [50]) and fraud detection (e.g. [29, 80]).

Time series data may be described in terms of several types of properties, such as trend, seasonality, cyclic, and irregular components [20]. Trend refers to an increase or decrease in the mean value of the series, e.g. showing an upward or downward trend in the stock prices of a company, whereas seasonality refers to a change in values for certain periods of time related to the calendar of the year, e.g. the demand of ice-cream. Cyclic behaviour describes a time series with predictable oscillations along time, and irregular components are defined as all other than the described components that are neither systematic nor predictable. A time series is considered stationary, if there is no systematic change in the mean or variance over a time period.

2.2 Forecasting

One of the main goals of time series analysis involves predicting the future values of the series. This process is frequently called forecasting. It has been widely used in many organizations where predicting the next value is crucial in making decisions. For example, energy generation organizations which use renewable resources to generate energy. These companies make use of the predicted values of consumption to generate sufficient energy from other sources in order to satisfy the demand [36]. Forecasting is usually focused on predicting the next value in a time series, but it is also used to predict multiple steps ahead in future, which is called as multi-step forecasting. But, difficulties can occur when predicting multiple steps ahead, due to the increased uncertainty [76]. Apart from predicting a single value from a time series, it is also possible to predict a range of values, since in certain domains one may be interested to find prediction intervals where the intended value lies between the lower and upper bound intervals [20].

2.2.1 Methods to Predict Future Values

We will discuss a few state of the art approaches of classical and machine learning methodologies used to forecast future values of a time series.

Classical Forecasting Methods

This subsection deals with the classical approaches to time series forecasting. Since forecasting involves predicting the future values in a time series, one simple method called the average method that finds the average of all values until that point and use this average as the forecast,

$$\hat{y}_{n+1} = \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{2.1}$$

where $\hat{y}_{n+1}$ denotes the prediction of time series for time $n + 1$.

Another simple method is the naive method that predicts the next value in the time series as the last known value,

$$\hat{y}_{n+1} = y_n \tag{2.2}$$

One of the most used classical methods for predicting univariate (using a single variable to forecast) time series is the Auto-Regressive Moving Average (ARMA) which combines two components: Auto-Regressive (AR) and Moving Average (MA).

In the Auto-Regressive method, the next value is predicted using the linear combinations of the past p observations at prior timesteps with error term ϵ_n and a constant term c [15].

$$y_n = c + \sum_{i=1}^{p} \phi_i y_{n-i} + \epsilon_n \tag{2.3}$$

where ϕ_i denotes the model parameters, p represent the order of the model and ϵ_n denotes the error.

On the other hand, Moving Average (MA) uses the linear combination of past errors to predict the next values.

$$y_n = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{n-i} + \epsilon_n \tag{2.4}$$

where μ denotes the mean of observations, θ_i denotes the parameters of the model, q represents the order of model and ϵ_n denotes the error.

Both the above methods are combined to produce effective prediction in ARMA(p,q). This is used in prediction of future values considering it as a linear combination of past values [20].

$$y_n = c + \sum_{i=1}^{p} \phi_i y_{n-i} + \sum_{i=1}^{q} \theta_i \epsilon_{n-i} + \epsilon_n \tag{2.5}$$

The variations of ARMA include ARIMA which includes an integration parameter to model the next steps in the sequence as linear functions of differenced observations. It is different from ARMA by a differencing pre-processing step, which is applied to make the time series stationary and then ARMA modelling is applied [15].

Another classical approach in forecasting is exponential smoothing. The forecasts are produced by a weighted average of the past observations with the weights decaying exponentially as it gets older [1]. The simplest in the class of exponential smoothing is Simple Exponential Smoothing (SES), which is suitable for data with no trend or no seasonal components [4]. In simple exponential smoothing, the weights of the recent past observations is greater than the older observations as given by the equation below,

$$y_{n+1} = \alpha y_n + \alpha(1-\alpha)y_{n-1} + \alpha(1-\alpha)^2 y_{n-2} + \cdots , \tag{2.6}$$

where α is the smoothing parameter which controls the rate at which the weights decrease and the value of α lies between 0 and 1. If α is small, more weight is given for observations in the distant past and if α is large, recent past observations gets more weight [4]. The value of α is chosen based on the importance of values in recent past. For example if $\alpha = 1$, it follows the naive method which takes the value of the recent observation, $y_{n+1} = y_n$.

Machine Learning Approaches

Apart from classical approaches to forecast time series, there are various machine learning approaches that are also used extensively in these tasks. The machine learning approaches aim to forecast the future values represented as the dependent feature. This dependent feature is forecasted based on certain features known as independent variables (input features). Frequently, the machine learning models use the past values of a time series as inputs and predict the future values. Since time series comprises of trend and seasonal components, certain machine learning methods incorporates this information as input variables to improve the performance of the model [14]. Several machine learning models have been compared with statistical methods to determine their performance. Makridakis et al., [47] discussed several machine learning models used in forecasting. Here we discuss a few approaches: Support vector regression (SVR), Recurrent Neural Networks and Long short term memory (LSTM).

Support Vector Regression (SVR) is the regression variant of support vector machines (SVM), which tries to find one hyperplane approximating the data. It finds a hyperplane in an N-dimensional space which best fits the data, where N represents the number of features. This hyperplane is used to predict the continuous output [64].

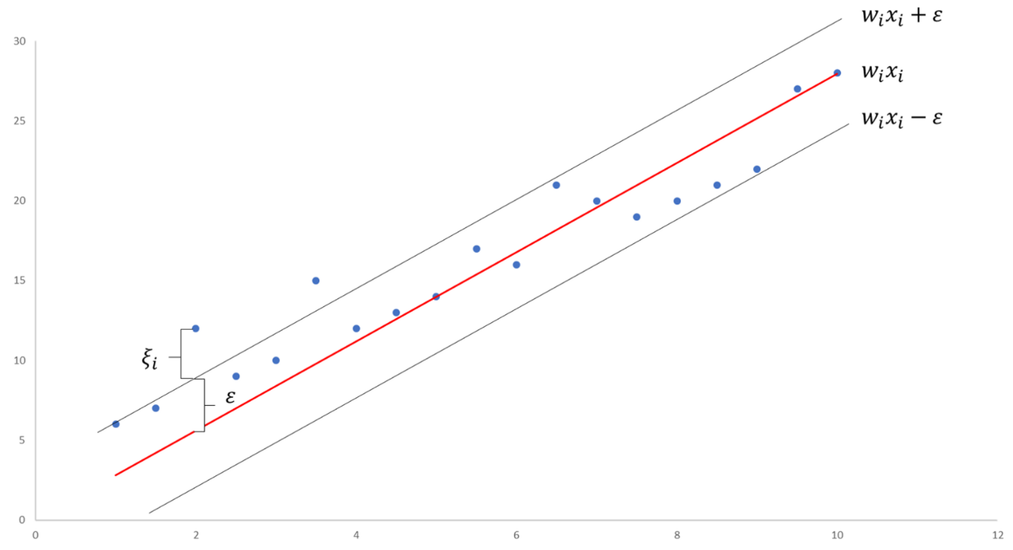


Figure 2.1: SVR with decision boundaries of distance ϵ
(figure taken from [66])

SVR has gained attention in a variety of time series problems including forecasting energy consumption [83] and financial time series [46], due to the structure risk minimization principle of SVM and great generalization ability. SVR deals with non-linear regression problems (where it is hard to find a hyperplane approximating the data) by converting them to linear tasks, by mapping them to a high dimensional space with the use of kernels [83]. The equation to solve for linear estimation function is given by,

$$y = f(x) = w \cdot \phi(x) + b \tag{2.7}$$

where w is the weight vector, $\phi(x)$ is the non-linear mapping function from input space to high dimensional space F and b is the threshold value [83].

SVR tries to find decision boundaries close to the hyperplane within an acceptable distance ϵ [66], as shown is Figure 2.1. The main objective is to consider the points within the decision boundaries as given by the $-\epsilon < w.\phi(x) + b < +\epsilon$. SVR tries to satisfy this condition but there may exist a few points outside the decision boundary. Thus, SVR tries to capture the errors for data points outside of decision boundary. We denote the deviation of points from the decision boundaries by ξ_i as shown in Figure 2.1. Thus, the main objective of SVR is to minimize the errors as much as possible, as given by the objective function,

$$Minimize \qquad \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}|\xi_i| \tag{2.8}$$

$$Subject\ to \qquad |y_i - w_i x_i| \leq \epsilon + |\xi_i| \tag{2.9}$$

where C represents the cost incurred due to each violation of error limit ϵ.

Sometimes, it becomes hard to find a hyperplane for non-linear problems to be solved in the original dimension. Thus, kernels are used to map data points from low dimensional space to higher dimension [83] and to make the optimization task feasible. Kernels reduce the need to calculate the complex dot products of high dimensional spaces [74].

Sagheer et al. [61] pointed out that Recurrent Neural Networks (RNN), is a class of neural networks that performs well with time series forecasting. RNNs recognize the past input sequences and help to predict the next values in the series. RNN stores the activation from each time step in an internal state of the network to provide a temporary memory. They retain information about the past and discover temporal correlations between events. However, it is hard for RNNs to learn long range time dependencies [53]. Thus, Long Short-Term Memory (LSTM) [42], that memorises the sequence of data from earlier stages, is used to forecast the future values [67]. LSTM has been widely used in many applications like natural language processing [70], speech recognition [62] and forecasting economic and financial time series [67].

Sagheer et al. [61] used an extension to LSTM called Deep LSTM (DLSTM), to forecast the petroleum production. DLSTM consists of multiple layers with each layer having multiple cells as shown in Figure 2.2, where the input at time t is denoted by X_t. The input X_t, is introduced to the first LSTM block along with previous hidden state $S_{t-1}^{(1)}$, where 1 represents the block number. The hidden state at time t, $S_t^{(1)}$ is computed and goes to the next time step and also to next LSTM block and so on until the last LSTM block gets compiled. Thus, it benefits from LSTM in a way that each layer processes and subsequently passes it on to other layer until the last layer produces the output [61].

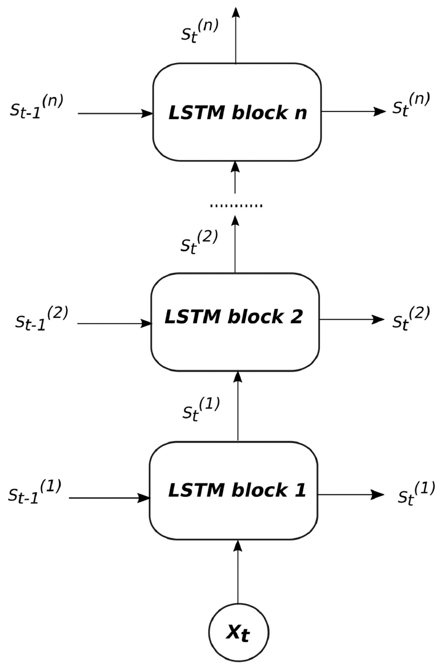


Figure 2.2: Deep LSTM Architecture (figure taken from [61])

2.3 Evaluating Forecasting Models

Model evaluation is a critical step to ensure the developed models perform well on new time series data [20]. Basically, we estimate the loss that the models will experience on unseen data. This section deals with the metrics used to evaluate the performance of the forecasting models and the estimation methods used to obtain reliable estimates of the values of these metrics.

2.3.1 Evaluation Metrics

The performance of predictive models is quantified by evaluation metrics. In this section, we will discuss some absolute and relative metrics that can be used to evaluate a forecasting model. First, we will deal with the absolute metrics used to evaluate the forecasting model. We start by introducing the mean squared error (MSE) metric used to evaluate the forecasting model, that calculates the mean of the squared difference between the predicted and actual value,

$$MSE = \frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n} \tag{2.10}$$

where $\hat{y}_i$ represents the predicted value, y_i denotes the actual value and n number of observations.

A variant of MSE called root mean squared error (RMSE) is also used to evaluate the forecasting model which calculates the square root of the mean squared error, that calculates the squared root of the mean of the squared difference between the actual and predicted values,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{2.11}$$

Another metric, similar to RMSE, used to evaluate the model by finding the difference between actual and predicted is root mean squared logarithmic error (RMSLE). RMSLE calculates the log error of the predicted and actual values and takes the mean square of it. Basically, it identifies the relative difference between actual and predicted values [2]. Thus, RMSLE treats big differences of actual and predicted in the same way as small differences, since it evaluates on relative differences. The formula to calculate RMSLE is given below,

$$RMSLE = \sqrt{\frac{\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y}_i + 1))^2}{n}} \tag{2.12}$$

It produces small value error even if the differences between actual and predicted are high. Thus, it differs from RMSE which produces a high error value, when the differences are high. RMSLE is used when there is no requirement to penalize for large differences.

Sometimes, to evaluate the forecasting models, mean absolute error (MAE) that calculates the mean of the absolute difference between the predicted and actual values is also used,

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n} \tag{2.13}$$

The absolute metrics discussed above, work better when used in a single time series problem, but is not as effective when evaluating multiple time series with different magnitudes [59]. To evaluate models on different time series problems with different magnitudes, relative metrics are preferred. Few relative metrics are discussed in this section, for example, relative mean percentage error (MPE) is calculated as,

$$MPE = \frac{100}{n} \cdot \sum_{i=1}^{n} \frac{y_i - \hat{y}_i}{y_i} \tag{2.14}$$

The relative mean absolute percentage error (MAPE) is formulated, as shown below,

$$MAPE = \frac{100}{n} \cdot \sum_{i=1}^{n} \mid \frac{y_i - \hat{y}_i}{y_i} \mid \tag{2.15}$$

Another metric used in evaluating the predictive models is mean absolute scaled error (MASE) [43], which is computed as,

$$\text{MASE} = \frac{1}{n} \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{\text{Loss}_{naive}} \tag{2.16}$$

where $Loss_{naive}$ represents the mean loss of the naive method in the data used to obtain the model and is denoted below,

$$\text{Loss}_{naive} = \frac{1}{n-1} \sum_{i=2}^{n} |y_i - y_{i-1}| \tag{2.17}$$

Another widely used metric in forecasting is R^2 or the coefficient of determination, that weights the individual scores by variance of the target variable [3]. R^2 is a statistical measure representing the proportion of variance of dependent variable as explained by the variance of independent variables [41]. Thus, if R^2 is 1, it means that, all the variance of the dependent variable can be explained by the variances of the independent variables. If R^2 is 0, it means that none of the variance of the dependent variable is explained by the variance of the independent variable and a score less than 1 means, only a few of the dependent variables' variance is explained by the independent variables. The formula is given below,

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{2.18}$$

where $\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$ represents the explained variance (explained sum of squares) and $\sum_{i=1}^{n} (y_i - \bar{y})^2$ denotes the total variance.

Another metric that is used in time series problems is the Theil Uncertainty coefficient (U), which provides a statistical measure of how well a time series of estimated values compares with the time series of observed values [5]. It provides a conditional

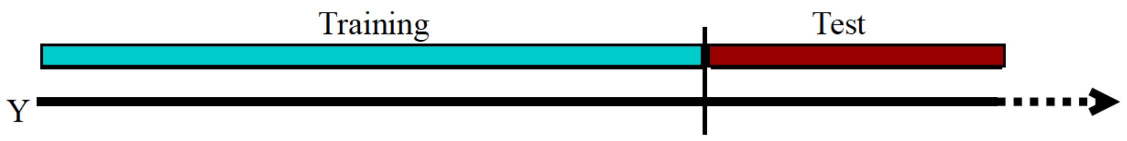


Figure 2.3: Simple holdout approach with train and test sets (figure taken from [20])

measure to find how well it predicts a variable with respect to another variable. Theil uncertainty helps to find the effectiveness of a statistical algorithm. It finds the correlation and not the correctness [6]. Thus, it does not penalize the algorithm for wrong predictions. The formula for Theil coefficient (U) is given below,

$$U = \frac{\sqrt{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}}{\sqrt{\sum_{i=1}^{n}(y_i - y_{i-1})^2}} \tag{2.19}$$

2.3.2 Estimation Methods

Different methods have been proposed to reliably estimate the values of the predictive performance metrics. The major groups of methods of predictive performance estimation in time series are Out-Of-Sample (OOS) and Cross-Validation (CV) [20].

Out-Of-Sample Methods

Out-Of-Sample (OOS) methodology tests the model using data that is not used for training the model, leaving a set of data exclusively for testing. Basically, it splits the dataset into two parts: one part for training the model and another for testing. The test set is held out to estimate the loss incurred by the model [20]. This approach is referred to as holdout and is depicted in Figure 2.3, where it shows the split between train and test data of a time series Y. For time series data we should make sure that the test set is posterior in terms of temporal order to the training set. This method does not make use of the whole dataset for performance estimation, but preserves the temporal order of data as test set is chosen posterior to the train set.

Tashman et al., [71] identified the different strategies within OOS, which includes different train/test split points and sliding windows. It selects random split points within each available window selected by the sliding window approach. OOS also

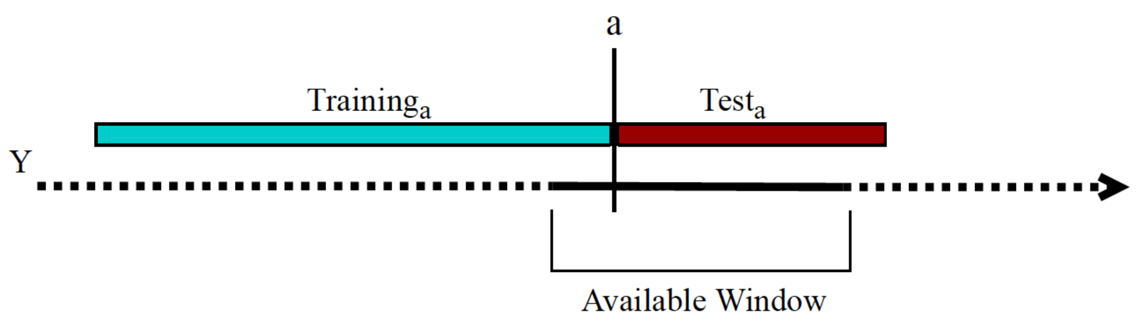


Figure 2.4: One iteration of repeated holdout approach with train and test sets chosen from available window (figure taken from [20])

includes a randomized approach called repeated holdout, where it randomly sub samples the data, consisting of different sets of train and test data for each iteration. This is illustrated in Figure 2.4, where it shows one iteration of repeated holdout [20]. Here, point a is chosen randomly from the available window, which marks the end of train data and the start of test data. This selection is repeated several times and the average scores on these repetitions will be the outcome of the estimation process.

Cross-validation Methods

Cross validation is one of the most commonly used methods when dealing with independent and identically distributed data [20]. K-fold cross validation is a method to randomly shuffles the data and splits into k-folds, where each fold is of size n/k, with n being the total number of observations and k denotes the total number of folds (blocks). This method makes use of the data efficiently with certain disadvantages for time series data, since it does not maintain the temporal order of the data when used in time series problems [8]. In K-fold CV, $k-1$ folds are taken for training and the remaining 1 fold is used for testing. This process is repeated for different combinations of k-folds and the loss is measured in each iteration.

A major distinction between OOS and CV is that, OOS model is never tested on past data [20].

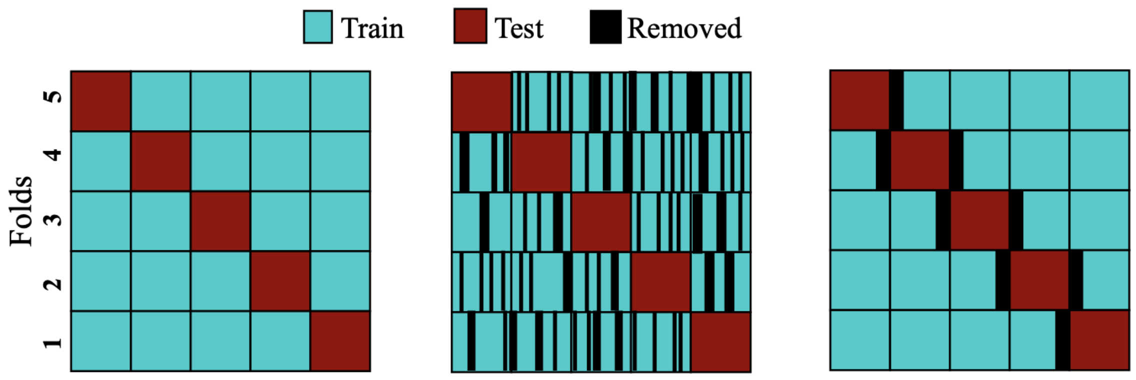


Figure 2.5: Variations on cross validation estimation procedures (figure taken from [20])

Variations on Cross Validation

The most common cross-validation works by initial shuffling of data and then dividing into k-folds; however, the Blocked Cross-Validation [68] ($CV - Bl$) is similar to usual CV, but there is no initial random shuffling. Thus, the temporal ordering of data within the block is preserved as compared to usual CV, but broken across the blocks. The first image of Figure 2.5 depicts the $CV - Bl$ scenario.

The modified cross-validation procedure [48] ($CV - Mod$) models the data by removing certain data from the training set, which is correlated with test data. As shown in the middle image of Figure 2.5, data in the training set is removed, for which the temporal range lies within test data. Thus, independence between train and test data is maintained in this method, but it leads to under-fit, since certain portion of data is removed.

The hv-blocked cross validation ($CV - hvBl$) [58] models data the same way as done by $CV - Bl$, but a certain amount of data adjacent to the test set is removed. As depicted in the third image of Figure 2.5, it removes the data before the start and after the end of the test data by a certain amount, thus creating a gap between train and test.

2.4 Activity Monitoring

In previous sections, we discussed the problem of forecasting time series. Activity monitoring is a related problem that also involves time series data. Activity monitoring involves identifying or anticipating specific events, that may occur in the future, which are related to the values of a time series. Specifically, an event is identified by a sequence of values in the time series crossing a certain threshold, which could be disastrous in many domains including healthcare, finance, environment, etc. The main goal of activity monitoring is to anticipate these future events in a timely manner [20].

The motivation for anticipating the events is to detect any problems as early as possible and to act in a timely manner to try to prevent or avoid them. Examples of real world problems involving anticipating events include health care monitoring systems, where an event is identified by a sequence of values of some critical health parameter crossing a certain threshold, that might be the main cause of any problem and anticipating those events would help in early diagnostics (for example, when a series of blood pressure values decrease below the threshold) [20]. In power generation from renewable sources, anticipating the solar or wind energy drop or increase in the future, would help the operators to take proactive measures to compensate the energy drop from the sources. It is also used in fraud detection systems to proactively react to frauds. Identifying those events as early as possible is challenging, but doing so, could save a lot of money and resources.

2.4.1 Challenges

The main challenge in anticipating the events of interest is that they are typically rare, which means, their frequency of occurrence is low and this creates serious difficulties to the prediction models as it becomes hard for the models to learn the rare events. Thus, it can be seen as a highly imbalanced learning problem [17]. It is quite similar to anomaly detection, which involves detecting values that deviate from the normality. Though both these problems occur in imbalanced domains, anomaly detection involves classifying the events as normal or abnormal. Whereas, activity monitoring mostly occurs in time series domains and the task is to anticipate the rare events which are of interest to many domains. Also, it is important to issue alarms in a timely manner before the events occur and the timeliness of alarms depends on the domain and the problem. There exists a certain timeframe, before which the alarm should be issued, which is useful in taking proactive measures or it becomes less useful, if the alarm is issued at times closer to the event [20].

2.4.2 Modelling Approaches

The two commonly used approaches involved in activity monitoring are:
Profiling [30], where the model is built using only normal activity of data excluding the abnormal cases. In this method, an alarm is issued for any activities different from the normal. This method could be used in cases where there is no proper definition of anomalies and anything other than normal is considered an anomaly. For example, it is used in fraud detection systems, where any activity other than normal transaction, causes an alarm to trigger.
Discriminating, where the model is built for learning how to discriminate anomalies from normal activity, i.e. as a classification problem. It uses information from a time series to classify the current state as leading to an anomaly or not. Thus, it issues alarms if found to be an anomaly, based on many predictor variables.

We now discusse a few methods designed for activity monitoring. The pioneering works of Weiss and Hirsh [77] include a model called Timeweaver, which uses a genetic algorithm to identify a diverse set of prediction patterns and generate rules that predict and generate alarms within specific time intervals using a sliding window. This method was used to detect telecom equipment failures using a set of alarm messages.

Another method for activity monitoring involves finding interesting patterns by monitoring events [30]. First, a set of rules is generated, which distinguishes positive events from normal and creates profiling monitors. Then, each monitor models and describes how far it is deviated from typical behaviour and generates alarms if there is enough evidence of unusual activity.

Stone and Veloso [69] approached activity monitoring as a layered learning problem and applied different layers to build up the model, as otherwise difficult to model from inputs to outputs. This method solves the problems by considering each problem as separate and solving one by one, rather than as a whole. They used layered learning for modelling robotic soccer, where it is distinguished into three layers: 1) Ball interception; 2) Pass evaluation and; 3) pass selection. The layers are applied in sequential manner where one layer affects the next. Using layered learning method, it is found to increase the efficiency of the decision-making system with high accuracy [69].

There are some situations where it is necessary to predict the class value of a sequence early in time when the full length is not available [51]. In this method, a probabilistic classifier is used to discover timesteps, where the accuracy for each class is defined by a threshold. In such a case, the class label is predicted only if the difference between the probabilities of top two classes is greater than the threshold. This is identified by comparing shapelets of different timesteps in the time series and it starts to classify only when the shapelets differs, thus reducing excessive classification check. This method is used to trigger a camera event based on early identification of birds [51].

Alternatively, there are some cases where it is required to transform into a forecasting problem in combination with activity monitoring. In such cases, it is formulated as a regression based approach [11], where the predictor values are forecasted initially and based on the forecasted values, it is decided whether to trigger an event or not. A classification model is also combined with regression to decide on the financial trading actions to buy, sell or hold an asset [10].

In aquaculture domain, the shellfish farms are monitored by predicting the amount of pollutants after high rainfall and high river flow [65]. It provides a decision support system for health authorities and aquaculture industries to close the farms in advance, if the predicted amount of pollutants is high. If unattended, the amount of pollutants increases, causing damages and loss to the industries.

2.4.3 Metrics on Activity Monitoring

As discussed from previous subsections, the main goal in activity monitoring is to anticipate the events in a timely manner rather than classifying them to be positive or negative for each sequence [30]. Some of the commonly used evaluation metrics in activity monitoring are recall, precision and F-score. Recall accounts for ratio of events correctly predicted as positive to the number of positive events in the dataset.

Let FN_m denote the number of false negatives in a test data set and let $\hat{T_m}$ represent the total number of those events correctly predicted by a model m [20]. The recall for the model m is given by the following equation:

$$R_m = \frac{\hat{T_m}}{\hat{T_m} + FN_m} \tag{2.20}$$

On the other hand, precision is defined as the ratio of number of events correctly predicted to the number of events predicted as positive by the model.

$$P_m = \frac{\hat{T_m}}{\hat{T_m} + FP_m} \tag{2.21}$$

where P_m denotes the precision, $\hat{T_m}$ is the number of events correctly predicted (true positives) and FP_m is the false positives predicted by the model m.

There is a known trade-off between precision and recall when choosing the model for prediction. F-score metric is used in order to maintain a balance between precision and recall for classifying the events [7]. F-score is used in imbalanced classification tasks where the events are rare, since precision would result in a very low value. F-score is low whenever precision or recall is low. Thus, it captures the events if any one score is low and produces a score close to the minimum score of precision or recall.

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.22}$$

2.5 Final Remarks

The dynamic nature of time series makes them one of the challenging research topics to discover knowledge from data. Although many state of the art methods have been proposed for forecasting the future values and predicting events, the problem persists to accurately predict them.

We discussed the basics of forecasting, activity monitoring, their methods, related works and evaluation metrics. The problems on uncertainty, class imbalance and predicting rare events were also discussed in this chapter. We will explore more about these topics when we describe our approaches in the next chapters.

Chapter 3

Case Study

We have discussed the background and state of the art methodologies in forecasting and activity monitoring in the previous chapter. In this chapter, we will focus on the application domain that drives the work of this thesis, providing necessary information and discussing the problems and challenges of the domain. We start by introducing the aquaculture domain and providing essential information on mussels and their role in assessing water quality in Section 3.1. This is followed by the data collection process in Section 3.2, where we provide information on the variables and features of the obtained dataset. Later, in Section 3.3, we describe information about micro closures in mussels, algae blooms and the relationship between them. Finally, in Section 3.4, we discuss the class imbalance problem in the dataset and the challenges in modelling this type of data.

3.1 Introduction

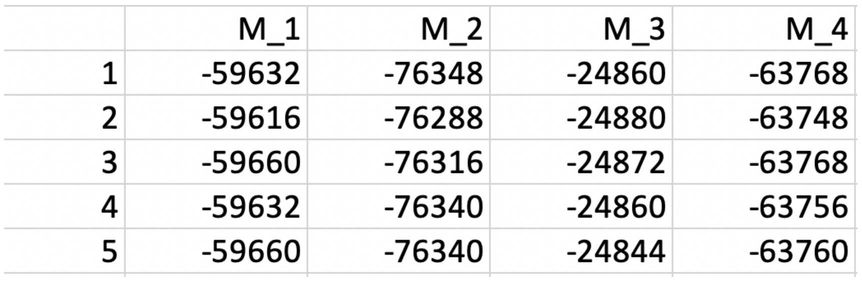
Aquaculture is one of the fastest growing food sectors in the world, accounting for more than half of the world's food production. It includes farming of fish, shellfish and aquatic plants under controlled conditions. As the world's demand for sea food increases rapidly, there is a need to cultivate sea foods in coastal areas with the advancements of technology [75]. With the rapid development of aquaculture, there is an intensive growth of cultivation. But there are many problems in aquaculture such as maintaining the water quality by removing toxins in the water and assessing the pollutants in the water that cause diseases in fishes. Thus, a proper monitoring and assessment system is required to maintain the growth of aquaculture and to meet the global demand in the consumption of sea foods.

Some researchers suggest that the quality of water can be assessed based on the amount of pollutants in the water due to high rainfall [65]. Researchers have found that water quality can also be measured by the amount of certain types of algae present in the water that is measured by the number of algae cells per millilitre of water. This measures the total algae count in that water environment which is used to detect the presence of algal blooms. As certain types of algae are harmful, that proliferates and causes harmful effects to the aquaculture species like dying of fishes. We will discuss algal blooms in the coming sections. The main objective of our thesis is to provide a solution to forecast the presence of algal blooms in advance without the need to measure algae by sampling water and analysing these samples in the Lab, saving time, energy and resources. This helps domain experts to treat the water, thus saving the aquaculture species.

Recent research shows that certain types of mussels, when placed in aquaculture farms, can be used to detect the presence of algae blooms in the water. They are a particular animal species from a set of families called bivalve molluscs, grown both in salt and freshwater ecosystems [52]. In this thesis, we will be working with the *Mytilus edulis* species, which are a set of blue mussels, used to detect algae in water. These mussels are extremely good at predicting different types of algae, however, we will focus only on the algae, *Alexandrium Tamarense*. It is a species of dinoflagellates known to produce saxitoxin, which is a neurotoxin that causes the human illness, clinically known as paralytic shellfish poisoning. We will be trying to forecast and assess the water quality in advance, based on the mussels' activity in the aquaculture farms. Thus, detecting the presence of algal blooms in advance will help to treat the water and in saving the aquatic species from dying.

3.2 Dataset and Features

The dataset for our thesis project was originally collected by a research group at the University of Quebec, Rimouski in 2019. The dataset consists of time series recordings of the measurement of mussel valve openings from the sensors attached to the mussels' valves which is recorded by a bio-monitoring system called valvometer. The dataset includes information of 4 mussels recorded at regular time intervals of 0.1 seconds. The values are measured in milli volts (mV). The values of mussel valve openings recorded by the valvometer range between -20000mV and -80000mV, where -20000 represents the voltage when valves are opened to the most and -80000 representing the voltage when the valves are fully closed. The valve openings for 4 mussels are



	M_1	M_2	M_3	M_4
1	-59632	-76348	-24860	-63768
2	-59616	-76288	-24880	-63748
3	-59660	-76316	-24872	-63768
4	-59632	-76340	-24860	-63756
5	-59660	-76340	-24844	-63760

Figure 3.1: First 5 rows of the original dataset showing mussels' valve opening voltage

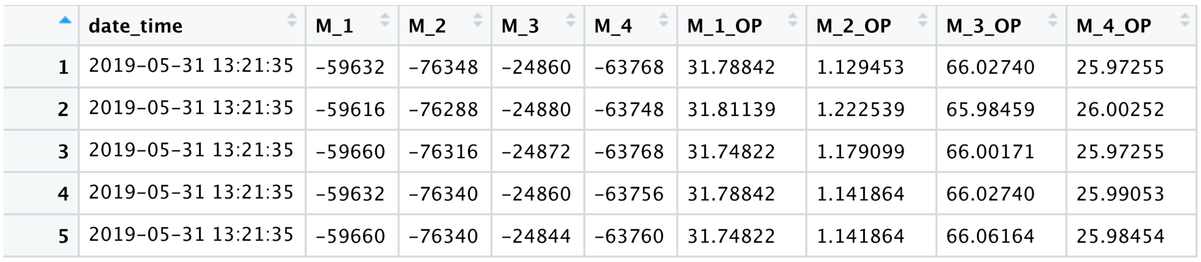
recorded for every 0.1s timestep during the period of 3 months (June - August), that accounts for nearly 77 million measurements.

In this section, we will discuss the collected datasets and explain the features derived from these data. First, we will discuss the mussel valve opening dataset. As mentioned previously, the dataset includes the information of the valve openings of 4 mussels. The provided dataset consists of four columns, each describing the measurements for one of the mussels. Figure 3.1 shows the first 5 rows of this dataset. The columns M_1, M_2, M_3, M_4 represent the voltage recorded from the valve opening of the four mussels. The five rows in the Figure 3.1 represent the information collected for the first 0.5 seconds, since the data collected for each row is recorded at a time interval of 0.1 second. Thus, the first column represents the timesteps and columns 2 to 5 depicts the voltage of valve openings of mussels measured by the valvometer, with the values ranging between -20000mV and -80000mV, as mentioned earlier.

As the meaning of the voltage information is not easily interpretable, it is hard to analyze the behavior of mussels with the voltage information. We have converted it into a scale that is easier to understand. Domain experts have found that voltages could be mapped to percentage of valve openings. This is done by mapping the voltage values to percentage by finding the differences between the voltage value and the minimum voltages of each mussel. Thus, -20000mV represents the valve is 100% open and -80000mV represents the valve is 0% open. For each recording, the percentage of the valve opening is calculated by the formula below,

$$Percentage_{ij} = \frac{V_{ij} - min(V_i)}{D_i} * 100 \tag{3.1}$$

$$D_i = max(V_i) - min(V_i) \tag{3.2}$$



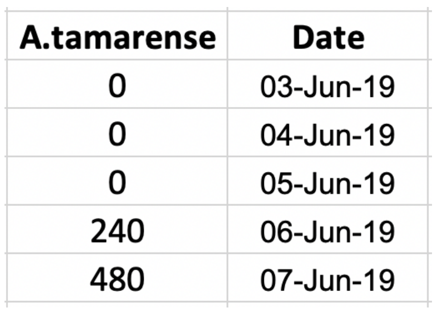
	date_time	M_1	M_2	M_3	M_4	M_1_OP	M_2_OP	M_3_OP	M_4_OP
1	2019-05-31 13:21:35	-59632	-76348	-24860	-63768	31.78842	1.129453	66.02740	25.97255
2	2019-05-31 13:21:35	-59616	-76288	-24880	-63748	31.81139	1.222539	65.98459	26.00252
3	2019-05-31 13:21:35	-59660	-76316	-24872	-63768	31.74822	1.179099	66.00171	25.97255
4	2019-05-31 13:21:35	-59632	-76340	-24860	-63756	31.78842	1.141864	66.02740	25.99053
5	2019-05-31 13:21:35	-59660	-76340	-24844	-63760	31.74822	1.141864	66.06164	25.98454

Figure 3.2: First 5 rows of the newly formed dataset showing mussels' valve opening percentage

where i represents each mussel, j representing each timestep, the voltage of mussel i at timestep j is given by V_{ij}, minimum of the mussel's voltage is given by $min(V_i)$ and D_i is the range of the voltages of mussel i.

Also, the date and time is calculated from the timesteps and added as a new column, since this will help in deriving extra features on the datasets based on date and time. Thus, after combining all these columns, the dataset looks like Figure 3.2, where it shows the percentage of each mussel and the date and time information of the recordings. The columns M_1, M_2, M_3, M_4 represent the voltage of the valve openings of the mussels and M_1_OP, M_2_OP, M_3_OP, M_4_OP denote the calculated opening percentage of the mussels' valves, respectively.

The amount of algae *Alexandrium Tamarense* in the water environment from where we extracted the voltage information on the mussels, is also measured on a daily basis during the same time period. The measured algae from the water environment ranges from 0 to 4480 algae cells per ml of water. Figure 3.3 shows the data of the measured algae during the first 5 days. The column *A.Tamarense* denotes the amount of algae per ml of water and *Date* represents the date of the measurements. In some days we have an algae count of 0, representing the absence of the algae *Alexandrium Tamarense*. From the Figure 3.3, we notice that during first 3 days, there is no algae in water, however, there seems to be more algae on June 6 (240 algae cells/ml of water) and June 7 (480 cells/ml). Since the recording of algae was obtained from June 3, 2019 till August 31, 2019, we have the algae count information for 90 days. The highest amount of algae *Alexandrium Tamarense* measured was 4480 algae cells per ml in August 18, 2019.



A.tamarense	Date
0	03-Jun-19
0	04-Jun-19
0	05-Jun-19
240	06-Jun-19
480	07-Jun-19

Figure 3.3: First 5 rows of the algae measurements in the water

3.3 Algae Blooms and Micro Closures

The main objective of this thesis is to predict future algae blooms, which are our target events. Algae blooms are events that occur when the number of algae cells per ml in water crosses a threshold, defined by the domain experts. For our thesis, we define the threshold to be 200 algae cells per ml of water, as advised by domain experts. If the algae cells in the water environment are greater than 200, we define the event as an algal bloom.

Researchers have found that micro closures of the valve opening of mussels can be seen as indicators of algal blooms. Micro closures are events that occur when mussels open and close the valves within a short time interval and the amount of opening and closing exceeds a threshold percentage as defined by the domain experts. The time interval to calculate the differences depends on the knowledge of domain experts.

Biologists believe that micro closures indicate the presence of algae in water and that more micro closures are indicative of more algae in the water. Thus, the amount of micro closures of the mussels can help us to find the presence of algal blooms.

In Figure 3.4, we have 4 micro closures, numbered from 1 to 4, determined using a threshold of $\geq 3\%$ from the openings and timesteps of 0.3s. The first micro closure occurs when there is an increase of 3% from 0.1s till 0.4s and then with an immediate decrease of 3% from 0.4s to 0.7s. The second micro closure occurs, due to the decrease in threshold value of 3% from 0.4s to 0.7s and with an immediate increase of value $\geq 3\%$ from 0.7s to 1s. The third micro closure occurs due to the increase in valve opening percentage of $\geq 3\%$ from 0.7s to 1s and with an immediate decrease of 3% from 1s to 1.3s and the fourth micro closure occurs due to a decrease of 3% from 1.8s till 2.1s and with an immediate increase of more than 3% from 2.1s to 2.4s.

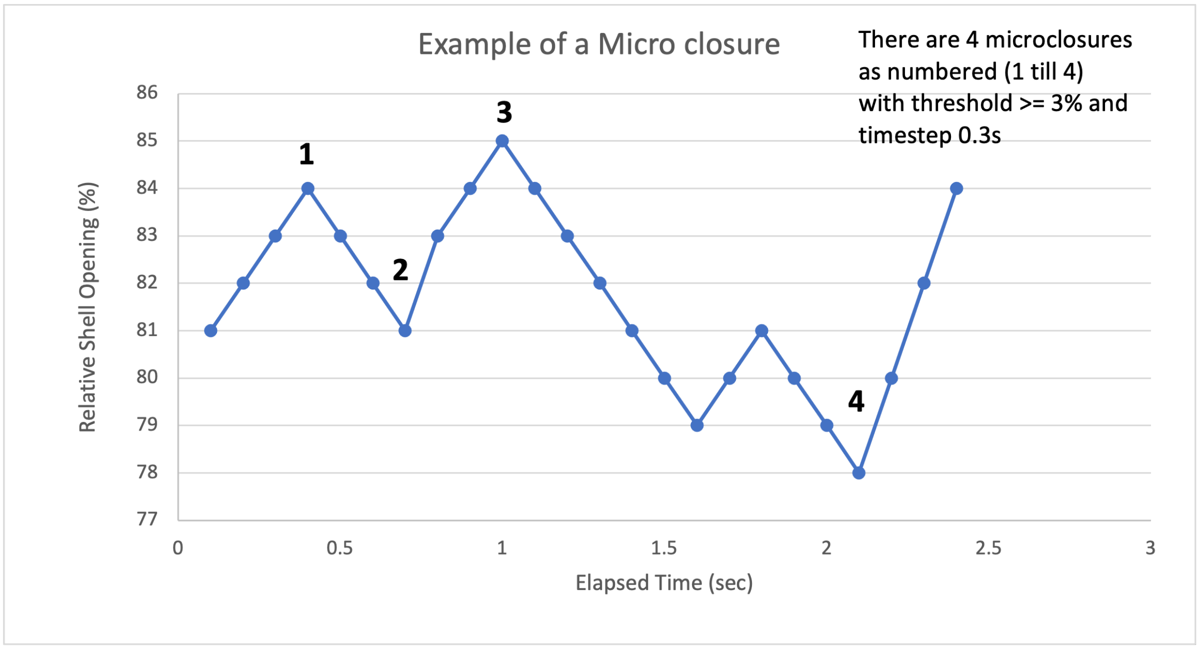


Figure 3.4: Example of micro closures

$$
Microclosure_t = \begin{cases} True, & \text{if } (v_t - v_{t-3} \geq 3 \text{ and } v_{t-6} - v_{t-3} \geq 3) \quad (3.3) \\ True, & \text{if } (v_{t-3} - v_t \geq 3 \text{ and } v_{t-3} - v_{t-6} \geq 3) \quad (3.4) \\ False, & \text{otherwise} \quad (3.5) \end{cases}
$$

3.4 Challenges in the Dataset

Having introduced the dataset in the previous sections, we discuss the challenges in the original dataset in this section. As the concepts of micro closures and algal blooms have been discussed in previous sections, there is a need to discuss the class imbalance problem. This problem arises when there is a large difference between the number of normal and target events. It occurs in many domains, where the target events are fewer than the normal events. For our thesis, we formalize the threshold for an algal bloom to be 200 algae cells per ml of water, as advised by domain experts. Thus, normal events are times when the algae count is < 200 and the target events are identified by times when algae counts are > 200. Distinguishing the events based on the algae counts, we found that there is a huge difference in the count of normal and target events in our dataset. As the target events are more important for the end users of this application, and they are rare ($< 5\%$ of the total events), we have what is known as a class imbalance problem [17].

Since the target events are rare, it becomes hard to learn them [17]. This is because machine learning models learn from the data we provide. This data is highly imbalanced, thus leading the machine learning models to focus the learning on the normal events, ignoring the important and target events as they have a more negligible effect on the metrics. This means we need some procedure that somehow bias the models towards the rare but more important events. Class imbalance problems and some solutions are discussed in Chapter 4.

Chapter 4

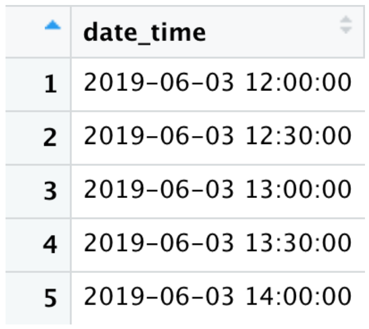
Methodology

As we have seen, in the context of our application domain, monitoring consists of the detection of algal blooms – our target events. The main objective of this thesis is to use Machine Learning models to monitor water quality and anticipate the presence of algal blooms without having to collect and analyse water samples from the aquaculture farms. This chapter describes the methods followed to achieve this objective. As discussed in the previous chapter, mussels can be used to infer water quality and thus detecting algal blooms. In Section 4.1, we analyze the mussels' behaviour with the goal of anticipating future algal blooms by creating new features for our forecasting models. Section 4.2 focuses on the proposed methodology for creating different models to predict algal blooms. We introduce the sampling approaches to solve the class imbalance problem of the resulting dataset in Section 4.3.

4.1 Feature Engineering

Given the description of the data in the previous chapter, the first task is to decide the target variable and the predictor variables to be used by the prediction models. This section deals with the manipulation of the data set to prepare it for building the models. We define a few predictor features used by the model to predict the events accurately. We also design the type of predictive task we solve to predict our target (algal bloom) events where algae count ≥ 200.

Our initial trials involved learning a forecasting model with the original data set consisting of just the valve opening percentages and the algae count. The results with this approach were not satisfactory. It becomes challenging for the model to learn the target events with just those features of the raw dataset. In this context, we have decided to create new features that could help in forecasting algal blooms by bringing new information as predictors. As we have stated earlier, micro closures in mussels' valve openings are crucial in detecting algal blooms, according to domain



	date_time
1	2019-06-03 12:00:00
2	2019-06-03 12:30:00
3	2019-06-03 13:00:00
4	2019-06-03 13:30:00
5	2019-06-03 14:00:00

Figure 4.1: Manipulation of dataset over 30-minute intervals

experts (aquaculture researchers at the University of Quebec). Thus, the new predictor variables are created based on the concept of micro closures discussed in the upcoming sections.

Since the main objective is to detect algal blooms that will happen in the future, we try to predict the events at specific time intervals ahead of the current time. These predictions will be obtained at a certain pace. In this thesis, we focus on predicting future algal blooms every 30 minutes, as advised by domain experts. We transform the original dataset, which was initially in a 0.1-second interval into 30-minute intervals starting from noon of June 3 till noon of August 31, as shown in Figure 4.1. The transformation of the dataset's timestep into a 30-minute interval is explained in the upcoming sections. The main task is to forecast the algal blooms in the future at a given time t. We make these forecasts for 3 different future time horizons: the next 30 minutes; 2 hours; and 4 hours. This decision was based on the opinion of the domain experts. This means that at time t we will have models that will forecast whether we should expect an algae bloom (algae count ≥ 200) in the next 30m, 2h or 4h.

4.1.1 Independent Features

Supervised machine learning models distinguish the features into dependent (target) and independent (predictor) features. As mentioned before, it was necessary to create new independent features to help the models in the task of anticipating blooms. These independent features are created based on the micro closures count over different timesteps.

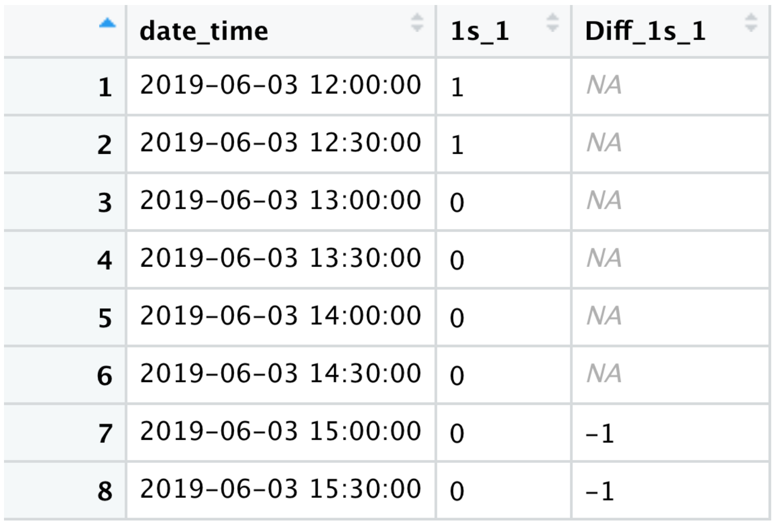
Differences on Micro Closure Counts

As describe in the previous chapter, micro closures are events that occur when the differences in the valve opening percentage over a short interval of time are greater than a threshold defined by domain experts. Since micro closures are used to identify the algae blooms, we create new features for the model using the total micro closure count of all mussels. We calculate the total number of micro closures over a time interval of 3 hours for every timestep (30 minutes), as suggested by domain experts. For example, we calculate the sum of micro closures of all mussels on June 3 from 9 am to noon (3 hours) for the features with different time intervals and thresholds on the noon of June 3 as explained below. As shown in Figure 4.2, we create a feature "1s_1" by calculating the sum of all mussels' micro closures over a time interval of 1 second and threshold of 1% within the 3-hour interval (9 am to noon). Likewise, we create many such features by calculating the total micro closures over different timesteps and thresholds within 3 hours interval for every 30 minutes.

We also create new features by finding the differences between the previously calculated micro closures count over a time interval suggested by the domain experts. For example, we calculate the difference of micro closure count over 3 hours on the feature "1s_1". As shown in Figure 4.2, we also create a feature called "Diff_1s_1" by calculating the differences of "1s_1" over 6 timesteps (3 hours). The values for "Diff_1s_1" starts at row 7 since we take differences over 6 timesteps and anything before is identified as "NA" . For example, at row 7 of "Diff_1s_1", we take the differences between the value at row 7 and row 1 on the column "1s_1". Thus, we create more such features by taking differences over 3-hour interval on the previously calculated features.

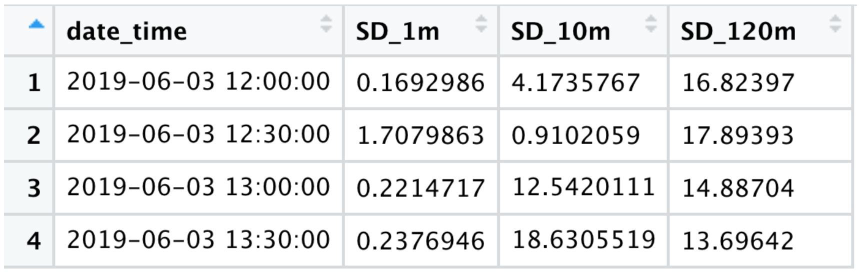
Standard Deviation of Valve Opening Percentage

We have also created features by calculating the standard deviation of the valve opening percentage over a certain period of time. Standard deviation measures the spread of a continuous variable. It provides information about the variation or deviation in the data. A low standard deviation indicates more values are around the mean value, whereas a high standard deviation represents a broader data spread. Thus, it captures mussels' activity over a certain time and useful in detecting any deviation



	date_time	1s_1	Diff_1s_1
1	2019-06-03 12:00:00	1	NA
2	2019-06-03 12:30:00	1	NA
3	2019-06-03 13:00:00	0	NA
4	2019-06-03 13:30:00	0	NA
5	2019-06-03 14:00:00	0	NA
6	2019-06-03 14:30:00	0	NA
7	2019-06-03 15:00:00	0	-1
8	2019-06-03 15:30:00	0	-1

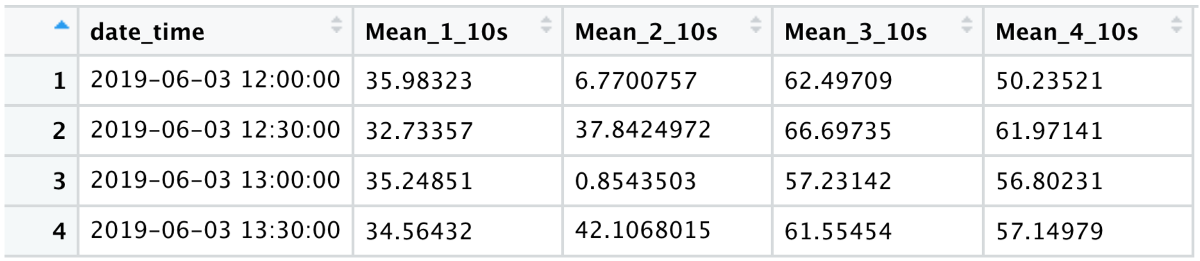
Figure 4.2: Example of differences in micro closure count over 1 second and threshold of 1%



	date_time	SD_1m	SD_10m	SD_120m
1	2019-06-03 12:00:00	0.1692986	4.1735767	16.82397
2	2019-06-03 12:30:00	1.7079863	0.9102059	17.89393
3	2019-06-03 13:00:00	0.2214717	12.5420111	14.88704
4	2019-06-03 13:30:00	0.2376946	18.6305519	13.69642

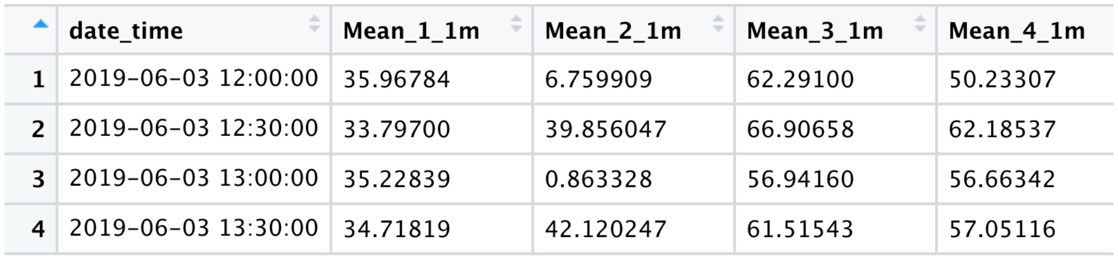
Figure 4.3: Subset of data showing 4 rows of the maximum standard deviation of the mussels over 1, 10 and 120 minutes

in the distribution of data. Since the dataset captures information on 4 mussels, we calculate the maximum standard deviation of the 4 mussels valve opening percentages. It provides a better understanding by capturing the highest deviation among the mussels. As shown in Figure 4.3, we calculate the maximum standard deviation of the 4 mussels over different time intervals. For example, we calculate the maximum standard deviation of the mussels' valve opening percentages over a period of the past 1 minute (11:59 am till 12:00 pm) on June 3, which forms the data for the column "SD_1m" at noon on June 3. Also, we create new such features by calculating the maximum standard deviation over different periods, namely of 10 and 120 minutes for the columns "SD_10m" and "SD_120m", respectively.



	date_time	Mean_1_10s	Mean_2_10s	Mean_3_10s	Mean_4_10s
1	2019-06-03 12:00:00	35.98323	6.7700757	62.49709	50.23521
2	2019-06-03 12:30:00	32.73357	37.8424972	66.69735	61.97141
3	2019-06-03 13:00:00	35.24851	0.8543503	57.23142	56.80231
4	2019-06-03 13:30:00	34.56432	42.1068015	61.55454	57.14979

Figure 4.4: Dataset showing 4 rows of the mean of 4 mussels over 10 seconds



	date_time	Mean_1_1m	Mean_2_1m	Mean_3_1m	Mean_4_1m
1	2019-06-03 12:00:00	35.96784	6.759909	62.29100	50.23307
2	2019-06-03 12:30:00	33.79700	39.856047	66.90658	62.18537
3	2019-06-03 13:00:00	35.22839	0.863328	56.94160	56.66342
4	2019-06-03 13:30:00	34.71819	42.120247	61.51543	57.05116

Figure 4.5: Dataset showing 4 rows of the mean of 4 mussels over 1 minute

Mean of Valve Opening Percentage

We have also included the mean valve opening percentage over specific time windows as another predictor. Compared to standard deviation that captures the data spread over time, the mean statistic captures the valve's typical opening percentage. It provides more predictive power into the data by capturing the average trend of the mussels' valve opening. We create new features by calculating the mean over a certain period for each mussel separately. As shown in Figure 4.4, we calculate the mean for each mussel over 10 seconds starting before the time in "date_time" column. The columns "Mean_1_10s", "Mean_2_10s", "Mean_3_10s", "Mean_4_10s" represents the calculated mean of 4 mussels over a 10-second interval. For example, we calculate the mean on the mussels' valve opening percentage from 11:59:50 am till noon (10 seconds) for the data at column "Mean_1_10s" and row 1. Also, we created a feature using mean over 1 minute time period as shown in Figure 4.5.

In summary, we have created a dataset with a series of features calculated for every 30 minutes interval using the raw mussels' data. Specifically, the following independent features are calculated on 30 minutes intervals: micro closure count over different timesteps and thresholds, the difference of micro closure count over 3 hour period, the maximum standard deviation of mussels' valve opening percentage over 1, 10 and 120 minutes, and mean of valve opening over 10 seconds and 1 minute for each mussel. Our approaches will use data sets containing these independent variables to learn models to predict the target feature explained in the next section.

4.1.2 Target Feature

Since our thesis's main objective is to forecast the events of algal blooms, we define the forecasting task as classifying the events as normal or a bloom. The normal events are situations when there are no algae or very few in the water environment, according to a threshold defined by domain experts. The target is to be able to forecast the critical events representing an algal bloom that should be taken care of to avoid serious problems in the aquaculture farms. Thus, the target feature is a binary variable with values bloom or normal.

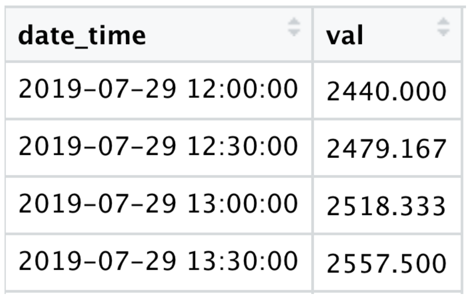
Estimating the Algae Counts Over 30 Minutes Periods

The raw data set with the algae counts consists of daily measurements over three months, as discussed in the previous chapter. Since our goal is to forecast algal blooms every 30 minutes, we need a more granular information on the algae counts. In this context, we have used a linear interpolation strategy to obtain a value of algae count for each 30m using the daily values in the available data set. The algae count for every 30-minute interval is calculated by linearly interpolating the algae counts over successive days. For example, in Figure 4.6, we linearly interpolate the algae counts for July 29 and July 30 with prerecorded values of 2440 and 4320, respectively. We include in our dataset these linearly interpolated algae counts over 30-minute intervals.

The formula to find the interpolated value is given by,

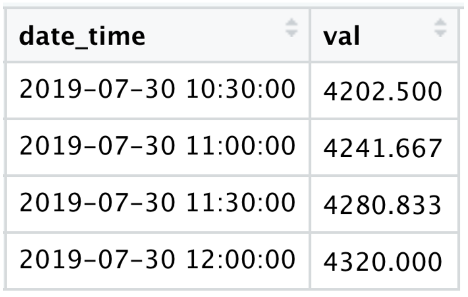
$$IV_i = V_t + [(i-1) \cdot \frac{V_{t+1} - V_t}{n}] \tag{4.1}$$

where IV_i represents the interpolated algae value at i-th timestep of the day with i=$\{1, 2, ..., n\}$, V_t and V_{t+1} represent the algae counts at current and the following



date_time	val
2019-07-29 12:00:00	2440.000
2019-07-29 12:30:00	2479.167
2019-07-29 13:00:00	2518.333
2019-07-29 13:30:00	2557.500

(a) Start sequence from noon of July 29



date_time	val
2019-07-30 10:30:00	4202.500
2019-07-30 11:00:00	4241.667
2019-07-30 11:30:00	4280.833
2019-07-30 12:00:00	4320.000

(b) End sequence till noon of July 30

Figure 4.6: Linear interpolation of algae counts over 30 minute intervals

day, n represents the number of timesteps. In our thesis, we interpolate values over 30 minutes as advised by domain experts. Thus, we define n to be 48, as there are 48 30-minute intervals in a day.

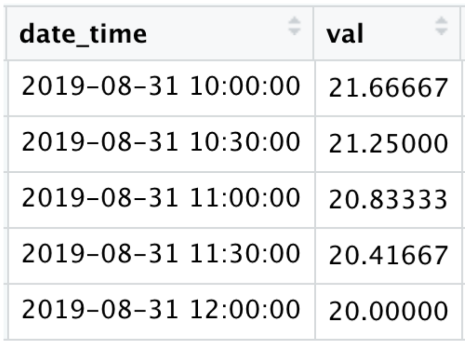
For example, as shown in Figure 4.6, interpolated value at 12:30 pm of July 29 is calculated by 2440 + (2 - 1) * (4320 - 2440)/48, which is equivalent to 2479.167 since it is the first timestep of the day.

Creating the Target Feature

As discussed in the previous section, the algae counts are manipulated by linear interpolation over 30-minute intervals. In our thesis, as advised by domain experts, we will be forecasting algal blooms at timesteps $t+1$, $t+4$, and $t+8$ which correspond to predict the events 30 minutes, 2 hours, and 4 hours ahead in the future, respectively.

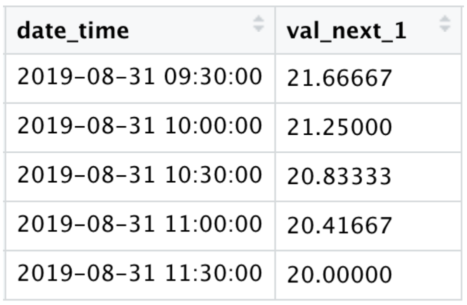
To predict the events of algal blooms at the future timesteps, we manipulate the dataset according to the future timesteps. For example, as shown in Figure 4.7, where (a) denotes the actual algae counts at each timestep, (b) represents the dataset to predict target variable at $t+1$ (30 minutes) timestep in future, (c) denotes the dataset to predict $t + 4$ timesteps (2 hours) in the future and (d) denoting the dataset for predicting target feature at $t + 8$ timesteps (4 hours) in future.

Having the linearly interpolated algae counts dataset for the forecasting task, the next step is to define the target variable. Domain experts suggest that the events are considered “normal” when the algae counts are less than 200 cells/ml and “blooms” when the algae count is greater than 200. Since our task involves detecting algae



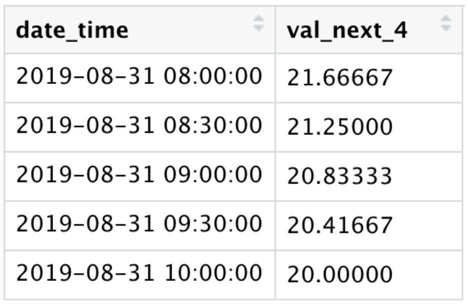
date_time	val
2019-08-31 10:00:00	21.66667
2019-08-31 10:30:00	21.25000
2019-08-31 11:00:00	20.83333
2019-08-31 11:30:00	20.41667
2019-08-31 12:00:00	20.00000

(a) Manipulation with original timestep t



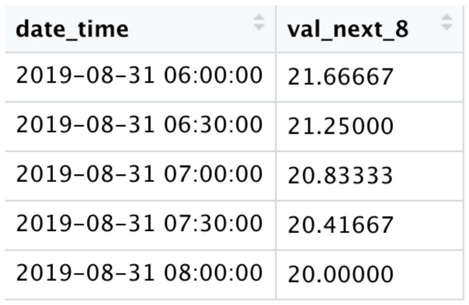
date_time	val_next_1
2019-08-31 09:30:00	21.66667
2019-08-31 10:00:00	21.25000
2019-08-31 10:30:00	20.83333
2019-08-31 11:00:00	20.41667
2019-08-31 11:30:00	20.00000

(b) Manipulation for $t + 1$ timestep (30 minutes)



date_time	val_next_4
2019-08-31 08:00:00	21.66667
2019-08-31 08:30:00	21.25000
2019-08-31 09:00:00	20.83333
2019-08-31 09:30:00	20.41667
2019-08-31 10:00:00	20.00000

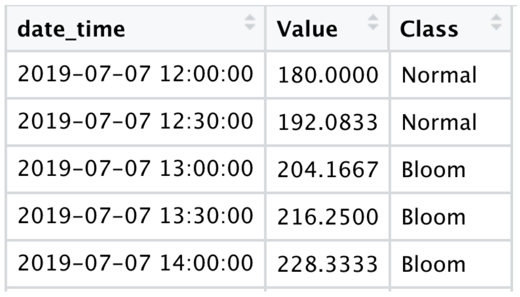
(c) Manipulation for $t + 4$ timestep (2 hours)



date_time	val_next_8
2019-08-31 06:00:00	21.66667
2019-08-31 06:30:00	21.25000
2019-08-31 07:00:00	20.83333
2019-08-31 07:30:00	20.41667
2019-08-31 08:00:00	20.00000

(d) Manipulation for $t + 8$ timestep (4 hours)

Figure 4.7: Linear interpolation of algae counts for predicting $t + 1$, $t + 4$ and $t + 8$ timesteps in the future



date_time	Value	Class
2019-07-07 12:00:00	180.0000	Normal
2019-07-07 12:30:00	192.0833	Normal
2019-07-07 13:00:00	204.1667	Bloom
2019-07-07 13:30:00	216.2500	Bloom
2019-07-07 14:00:00	228.3333	Bloom

Figure 4.8: Dataset showing 5 rows with Value and Class column

blooms, our target variable represents the events to be normal or a bloom. Thus, the target variable is defined as a 2 class feature depicting algae blooms' presence or absence.

We obtain the target variable values using the interpolated dataset by classifying it as "normal" if the algae counts at the future timesteps are < 200 and "bloom", if it is ≥ 200. This forms the dependent target variable for our forecasting models. As shown in Figure 4.8, since the values for the first two rows are less than 200, the target class column is classified to be "Normal" and for other rows, the values are greater than 200; thus, classified as "Bloom". Therefore, it is modeled as a (binary) classification task involving the detection of the events as normal or bloom at the future timesteps, namely $t + 1$, $t + 4$, and $t + 8$.

4.2 Proposed Forecasting Methodologies

In this section we discuss the methodology followed to forecast algae blooms using different machine learning algorithms. The basic idea in detecting the events is to approximate a function h that maps a set of input features U to a target feature v, which in our case is a binary variable indicating whether we anticipate a bloom or not.

Our thesis's main objective is to produce a forecasting model that predicts future algae blooms. As researchers found out that micro closures in the mussels' valves can be used to detect the algal blooms in water, we have used several variables based on the micro closure counts as predictors. These features represent the input variables U for the various machine learning models built to predict the binary target variable v. We do not include the algae counts measured at past time-steps as predictor features since the main objective of this work is to use mussels' valve readings to avoid having to measure algae counts, which is a time consuming and expensive process. Thus, we have designed the forecasting task to predict future algae blooms using features based on the micro closures in the mussels' valves without the past algae counts as predictors.

In Chapter 2 we discussed various models that can be used to forecast the next values in a time series. In our thesis, we use the following models: SVM, Random Forest, and XGBoost. This selection was based on the fact that these models provided good performance to forecast algal blooms and are relatively easy to tune the model's parameters compared with, for instance, Neural Networks.

4.2.1 Support Vector Machines (SVM)

Support Vector Machine (SVM) [27] is a supervised machine learning model used in classification and regression tasks. SVM's primary objective in a binary classification task is to find the best hyperplane separating the classes. It finds a hyperplane in an N-dimensional space which best fits the data, where N represents the number of features. This decision boundary separates the data points into different classes representing the dataset. These hyperplanes provide a linear separation of the data points in a binary classification problem.

There may exist many hyperplanes separating the data points. However, the hyperplane that maximizes the distance from the decision boundary to all classes' data points is chosen. Thus, the ideal hyperplane is chosen such that the distance from the hyperplane to the nearest data point of all classes is the largest - known as the maximum margin hyperplane.

The dimension of the decision boundary depends on the number of features in the dataset. For example, if there are two features, the decision boundary is a line, and for three features, the decision boundary becomes a 2-dimensional plane [35]. Figure 4.9 depicts the scenario of hyperplanes in 2D and 3D space.

The data points closer to the hyperplane are called support vectors [35]. These support vectors influence the hyperplane's orientation, and removing some of them would change the hyperplane's position. As the model tries to maximize the margin of the data points from the hyperplane, support vectors play an essential role in

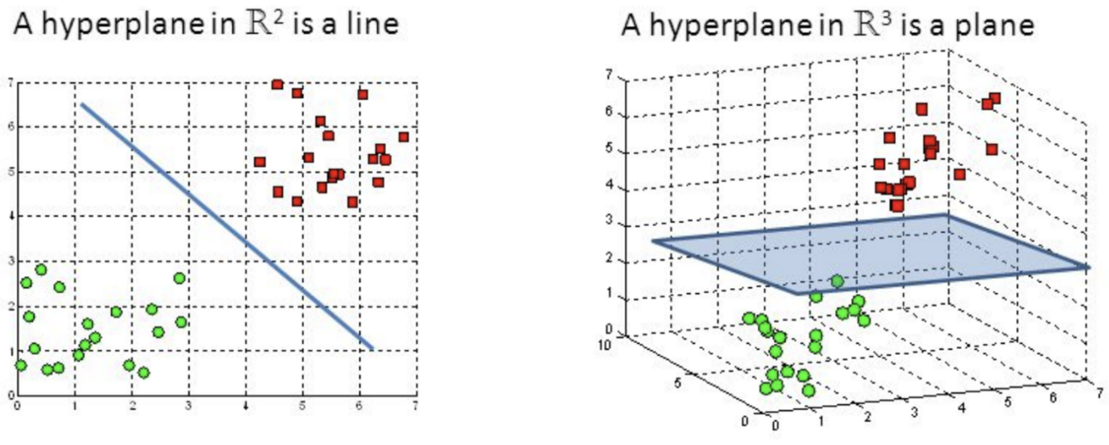


Figure 4.9: Hyperplanes in 2D and 3D (figure taken from [35])

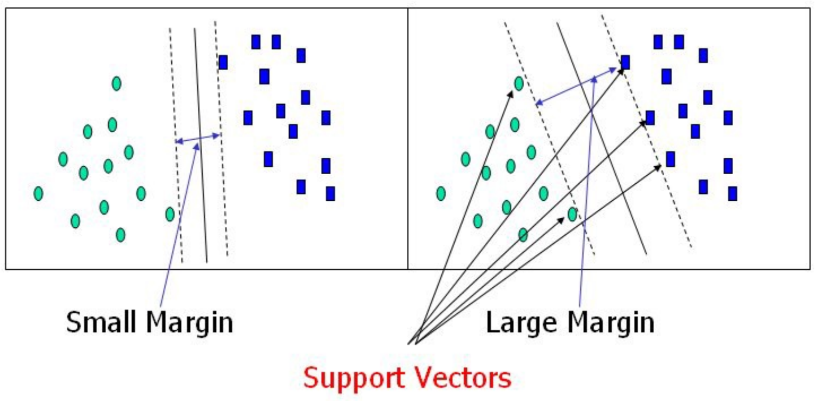


Figure 4.10: Support Vectors and Margin (figure taken from [35])

achieving it. Figure 4.10 depicts this scenario, where the ideal hyperplane is chosen to maintain a large margin compared to a hyperplane with a small margin for the same dataset.

The separating hyperplane in a SVM model is given by

$$f(x) = w \cdot x + b \tag{4.2}$$

where $w \cdot x$ is given by $\sum_{i=1}^{p} w_i x_i$ with i denoting the number of data points and w represents any candidate solution [74]. The distance from the support vector to w is given by $1/||w||$ and the size of the margin separated by both classes is given by $2/||w||$.

A constrained optimization problem exists where the objective is to maximize the distance from margin by minimizing $||w||$ subject to $y_i(b + w_i x_i) \geq 1$, $\forall i$, where y_i is given by,

$$y_i = \begin{cases} +1, & \text{for } x_r \quad (4.3) \\ -1, & \text{for } x_l \quad (4.4) \end{cases}$$

The constrained optimization problem can be solved by Lagrangian multiplier method [74]. The main objective is to maximize the following expression with respect to α_i representing set of positive Lagrange multipliers with i=1,2,...,n,

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{4.5}$$

where, $\alpha_i > 0$ correspond to support vectors

In real-world scenarios, the data points are not always linearly separable. It becomes difficult for the model to separate the classes in a non-linear dataset by a line represented in 2 dimensions, as shown in Figure 4.11. However, we can classify the data represented in low dimensions by transforming them to a high dimension where we find a hyperplane to exist, separating the samples. For example, in Figure 4.12, the data is transformed into a 3-dimensional space where a hyperplane separating the 2-class data is found. Thus, the data points from non-linear separable datasets are mapped into higher dimensional spaces to find a hyperplane separating the data.

The solution to the optimization problem in Equation 4.5 involves computationally complex dot products on high dimensional spaces. These dot products can be replaced by kernel functions evaluated in the original smaller space to provide a simpler calculation as given below,

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) \tag{4.6}$$

where ϕ denotes the mapping function that is used to transform data points in original space (low dimensions) to higher dimensions.

Thus, it reduces the need to calculate the complex dot products of high dimensional spaces [74]. There are different types of kernels, including polynomial, Gaussian, Radial Basis Function (RBF) and Sigmoid. Figure 4.12 shows an RBF kernel that maps data points from 2-dimensions to 3-dimensions to classify them [13].

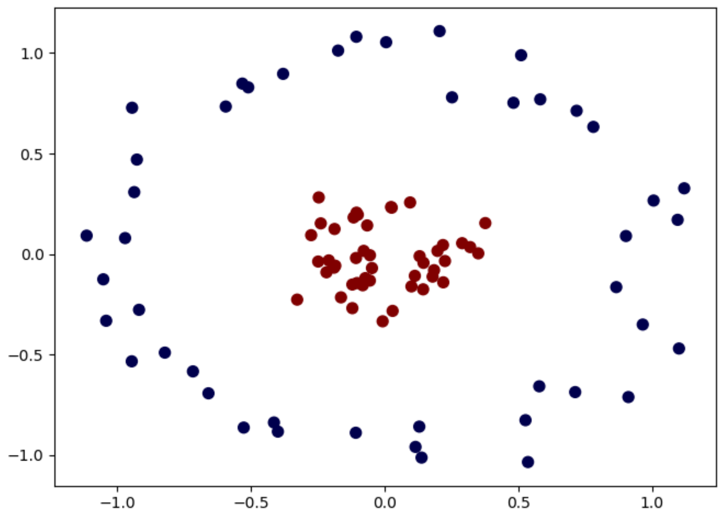


Figure 4.11: Non-linear separable dataset (figure taken from [13])

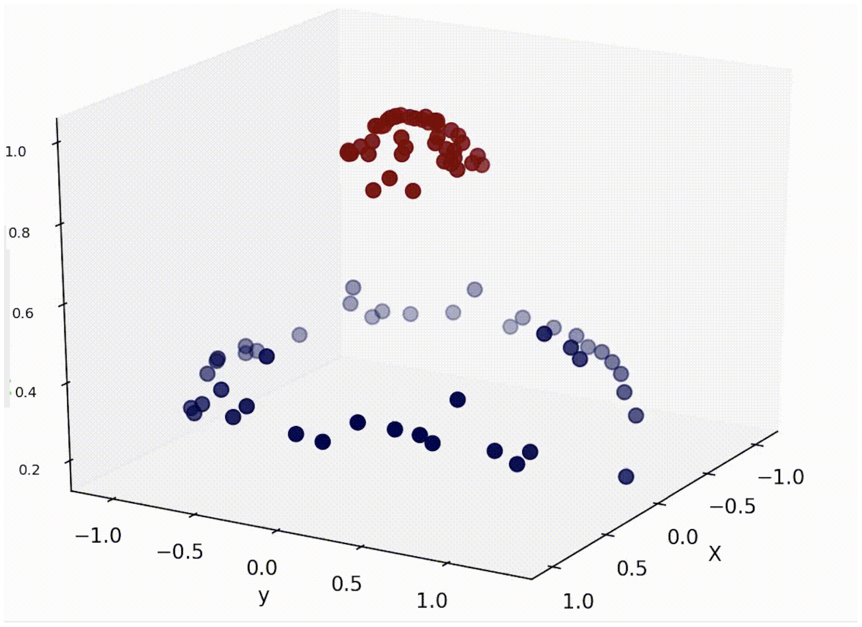


Figure 4.12: Finding Hyperplane in 3D space by RBF kernel (figure taken from [13])

There exists some trade-off in choosing the best hyperplane to classify the data points which are controller by certain hyperparameters discussed in next chapter.

4.2.2 Random Forests

Random forests are one of the most prevalent machine learning algorithms used in classification and regression. It is an ensemble tree based learning algorithm that constructs many decision trees to obtain better prediction. Random forest is different from decision trees as it is made up of various decision trees, each built differently to ensure a certain level of diversity among them, as discussed in the sections below. Since our objective is to forecast algal bloom events, we focus on the use of random forests for classification tasks. We will discuss the decision tree classifier in brief before looking into the random forest. As the name implies, the decision tree algorithm finds decisions at each step, called the splitting node, and the data is split continuously based on the features [56].

Decision tree consists of 3 components:

1. **Nodes:** test values of certain feature.
2. **Edges:** outcomes from the test node and connects to next node or leaf.
3. **Leaf Nodes:** predicts the outcome, representing the class.

A simple decision tree classification is shown in Figure 4.13, where the tree classifies the fitness of a person based on age, eating habits, and workout plan. In the first step, it divides the data into two groups, splitting at node "age" with values < 30 and > 30, which forms the initial splitting node. Then, it finds the next splitting node based on the feature "eating pizzas" within the group $age < 30$. It splits the data based on eating habits into two groups and classifies them as unfit or fit. On the other hand, for data with $age > 30$, it splits into two more groups based on "exercises in the mornings" and classifies them as fit or unfit. Thus, the decision tree splits for features until it finds groups with similar classes, or it stops based on certain conditions explained below.

The stopping conditions for a decision tree are:

- until all groups are pure (the groups having a common class)

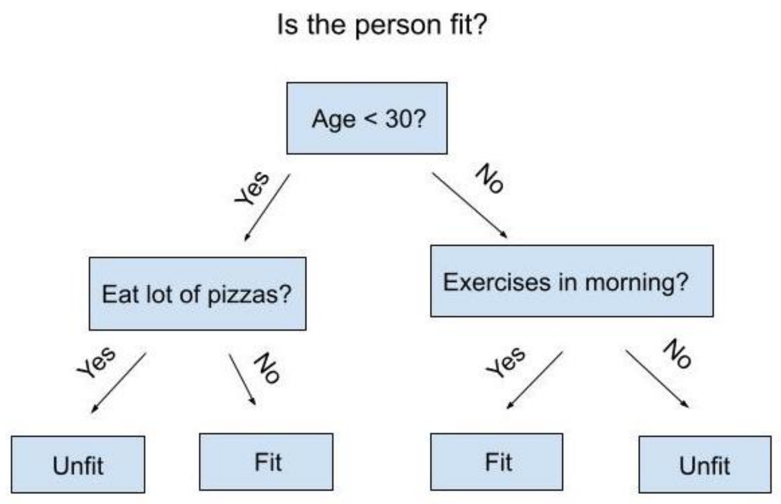


Figure 4.13: Decision Tree classification (figure taken from [21])

- given a limit on the depth of the tree, to avoid overfitting [56]

Thus, the decision tree classifier classifies the data by creating subgroups and maintaining similar classes within each group, but different between groups [81].

We have introduced the working principle of decision trees, which forms the basis of random forest classification. As discussed, Random Forest is a ensemble tree-based classification algorithm used in classification and regression tasks [18]. Figure 4.14 shows a simple random forest classifier with nine different decision trees that operate as an ensemble, where each decision tree predicts the output to be either 0 or 1. The total predictions of the decision trees are six 1s and three 0s. Since the task is classification, the random forest classifier predicts the overall result to be 1, based on the majority votes.

The low correlation between the decision trees in the random forest helps predict the correct output [18]. Thus, the random forest has a high chance of making correct predictions, since the errors in a single tree do not affect other trees.

Random forest ensure diversity among the different trees using the following methods:

- **Bagging**, where each tree is obtained using a different training set, namely by sampling with replacement (bootstrap sample) the rows of the original dataset [57]. The final classifier is formed by aggregating individual classifiers and

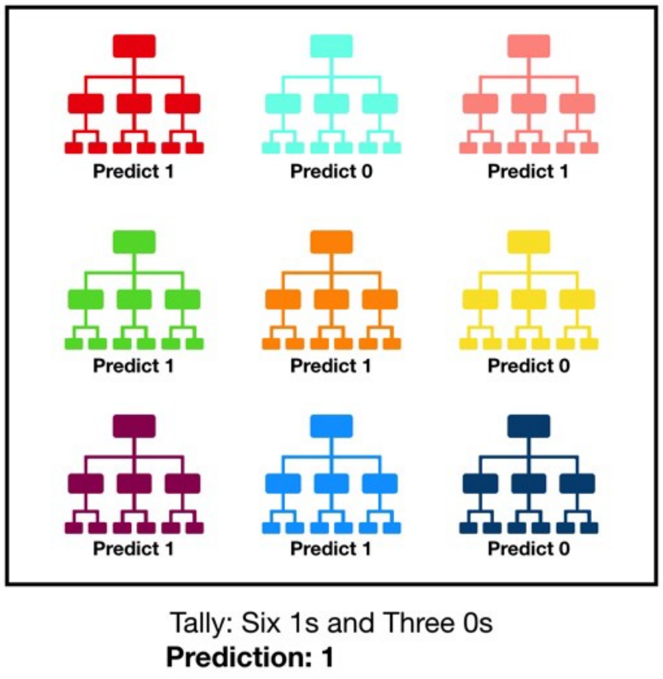


Figure 4.14: Random forest classification (figure taken from [81])

classifies the sample based on the majority votes of the individual classifiers.

- **Feature Randomness**, where for each tree node the best split test is selected from a random subset of the features in the original data set [18].

To summarize, random forests combine different decision trees generated using bagging and feature randomness. The parameters to improve the performance of the random forest are discussed in the next chapter.

4.2.3 XGBoost

XGBoost (Extreme Gradient Boosting) is another machine learning algorithm used in regression and classification tasks. XGBoost belongs to a family of boosting algorithms and uses Gradient Boosting Machine (GBM) framework at its core. We will discuss the basics of boosting and gradient boosting before discussing XGBoost. Boosting is a sequential and iterative approach that combines many models [31]. In a classification task involving boosting, the classification algorithm is iteratively applied to the training data, each time using different weights on the samples. Boosting works by sequential modeling and not by training models separately and puts more weights for instances that were predicted wrongly in the previous iterations. Thus, the main objective of boosting algorithms is to correct the mistakes caused by previous models and to produce a better prediction model.

Gradient boosting is a type of tree boosting algorithm that trains models in a sequential and additive process. It identifies the shortcomings of the previous models using gradients in the loss function. It minimizes the errors at each training data point evaluated for the model [32]. The gradient boosting algorithm's working hypothesis is the following: first, a base learner selects a random sample of the training dataset and updates the model parameters for the current iteration based on the previous iteration's gradients in the loss function; then, it trains based on the different random sub-samples of the training data and updates the model parameters in each iteration.

XGBoost [25] is a scalable tree boosting algorithm that uses a gradient boosting machine. The main goal is to minimize the regularized objective as given by,

$$L^t = \sum_{i=1}^{n}[l(y_i, \hat{y_i}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \tag{4.7}$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2 \tag{4.8}$$

where, $g_i = \partial\hat{y}^{t-1} l(y_i, \hat{y_i}^{(t-1)})$ and $h_i = \partial^2\hat{y}^{t-1} l(y_i, \hat{y_i}^{(t-1)})$ denotes the first and second order gradient statistics on the loss function $l()$, respectively. The prediction at the i-th instance and $t-1$ iteration is given by $\hat{y_i}^{t-1}$, $l()$ is a loss function that denotes the difference between $\hat{y_i}^{t-1}$ and y_i. f_t represents independent tree structures that are added to minimize the regularization objective and Ω penalizes the complexity of the model. Also, λ denotes the regularization parameter in Equation 4.8, T represents the number of leaves in the tree, γ denoting the complexity cost, and the leaf weights are represented by w.

To summarize, XGBoost is a machine learning approach based on a tree-based ensemble technique that works by combining models sequentially and by correcting the previous models' errors.

4.3 Handling Class Imbalance

Many supervised machine learning models assume the dataset to be balanced with equal proportion of the target class values [12]. But, this is not the case in many real world applications, where a large number of samples belonging to one class outnumbers the samples in another class. This is known as class imbalance. It becomes challenging for the machine learning models to generalize the data and classify correctly when the end-user is particularly interested in the less frequent class. Thus, it becomes hard to identify samples belonging to the minority class affecting the performance of machine learning models on the minority class.

Analyzing the data set of mussels and algae, we found more normal events than algal blooms. This means that the class values in our dataset are not equally represented, and thus we have a class imbalance problem because we are particularly interested in forecasting the less represented class [24], the blooms. Most machine learning models are not appropriate to handle class imbalance datasets. One way to solve a class imbalance problem is to change the distribution in the original data by oversampling or undersampling the data to balance the classes of normal and target events. We will discuss a few resampling methodologies used in our thesis in the sections below.

4.3.1 Random Under-Sampling

Random under-sampling is one of the simplest methods used to re-sample the dataset [16]. It addresses the class imbalance problem by randomly removing some samples of the most represented and least important class until the desired class distribution is achieved [12]. Though under-sampling increases the models' sensitivity towards minor classes, it can remove potentially useful examples for the task of discriminating among the class values. This method produces better results if the majority data points are close to each other, and removing a few of them does not result in a significant loss of data.

4.3.2 Random Over-Sampling

Random over-sampling is another resampling method that works by introducing replicas of the examples of the least represented class. It creates random duplicate samples of the minority class and adds them to the training dataset. Thus, it helps balance the distribution by random replication of minority data [12]. Random over-sampling provides a better approach to solve the class imbalance problem without removing any data. However, random over-sampling increases the size of the dataset, which may be computationally expensive if the original data set is already too large [55].

4.3.3 SMOTE

SMOTE (Synthetic Minority Over-sampling TEchnique) is a combination of over-sampling and under-sampling approaches, where minority samples are synthetically created and added to the dataset [24], while majority class examples are randomly removed, based on a specified percentage. The procedure used to introduce new minority class examples is different from other over-sampling methods that essentially replicate examples. Alternatively, SMOTE creates samples by performing interpolation on the existing samples of minority class. The over-sampling of the minority data occurs by introducing synthetic samples that come along the line segment between the chosen minority sample and any/all of its K nearest neighbors. The data points from K nearest neighbors are randomly chosen based on the amount of sampling required [16].

The method used by SMOTE case generation step is as follows [24]: initially, the difference between the feature vector (sample) and its neighbor is calculated; then it is multiplied with a random number chosen between 0 and 1 and is added to the sample under consideration. Thus, an arbitrary point is selected on the line segment between an existing minority class case and its neighbor, representing the synthetic sample. Therefore, it effectively creates synthetic data samples on the line segment between the two chosen minority data points. As shown in Figure 4.15, a neighbor is chosen among the sample's K neighbors, and synthetic data is introduced along the sample's direction. Thus, it introduces new synthetic data by interpolating the sample features [16].

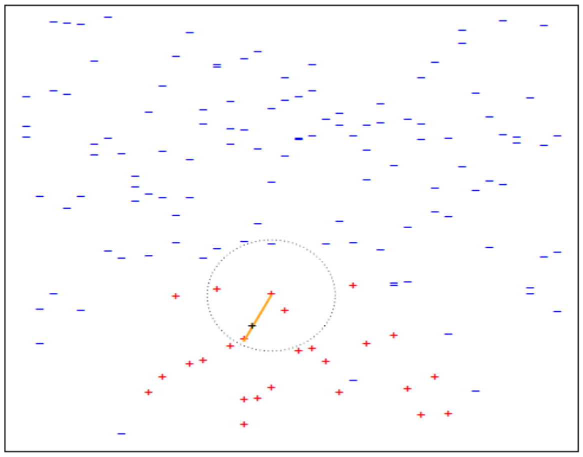


Figure 4.15: Synthetic data points generated by SMOTE algorithm (figure taken from [16])

Chapter 5

Experimental Evaluation of the Approaches

In the previous chapter we presented several machine learning models (SVM, Random Forest, and XGBoost) that can be used to predict algal blooms. We have also discussed a few sampling methods such as Random Under-Sampling, Random Over-Sampling, and SMOTE to balance our dataset. This chapter describes the experimental evaluation of these methods and discusses the obtained results.

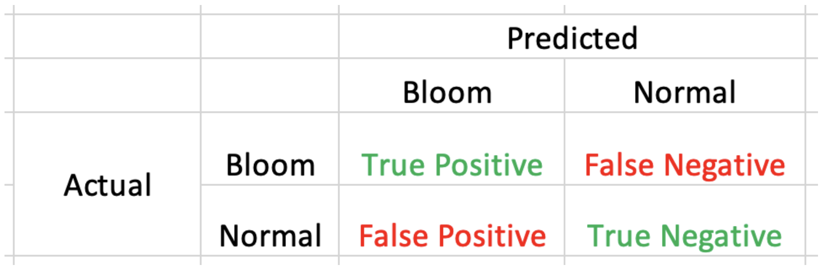
5.1 Experimental Settings

Having discussed the machine learning models in previous chapter, we discuss the experimental settings and parameter tuning to achieve better performance with the models in this section. The experiments are carried out in the R programming language and we use the packages $e1071$ [49], $randomForest$ [45], *xgboost* [26] for the machine learning models SVM, Random Forest and XGBoost, respectively. For the sampling methods Random Under-sampling, Random Over-sampling and SMOTE, we use the functions $RandUnderClassif$, $RandOverClassif$, $SmoteClassif$ respectively from the R package UBL [16]. We use the R package $performanceEstimation$ [72] to carry out the experiments. In the sub-sections below, we describe some of the parameters of these functions and how we have set them in our experiments.

5.1.1 Evaluation Metrics

The main objective of our thesis is to develop models to forecast algal blooms. We have formalised this task as a classification problem in which for each test case we classify it as a bloom or not. Given a set of test data, we predict their class labels using the trained models. The models are evaluated based on the errors of their predictions for the test data, using the metrics discussed below.

The predictions of a classifier for this type of binary classification tasks can be depicted by a confusion matrix like the one shown in Figure 5.1, where the positive



		Predicted	
		Bloom	Normal
Actual	Bloom	True Positive	False Negative
	Normal	False Positive	True Negative

Figure 5.1: Confusion Matrix

events represent the algal blooms and the negative represent the normal events.

Accuracy

Accuracy (or its complement error rate) is one of the most used evaluation metrics in a classification task, which measures the proportion of test cases predicted correctly. It can be calculated from the confusion matrix of Figure 5.1 as follows,

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{5.1}$$

where, TP: True Positives, TN: True Negatives, FN: False Negatives and FP: False Positives.

Although accuracy is one of the most used metrics in a classification tasks, it should not be used when dealing with rare events [77]. It would produce high accuracy even if the models predicted all the events to be normal and missed all the rare events. For example, if there are 97 normal events, 3 target events, and the model predicts all the events as normal, it still produces an acceptable accuracy score of 97%. But the model fails to capture the rare target events. Thus, we do not use accuracy when evaluating the performance of our forecasting models to detect the rare algal blooms. The following metrics are preferred when evaluating a forecasting task that includes rare events.

Precision

Precision measures the proportion of positive events predicted by the models that are indeed true positives. As discussed in Chapter 2, it measures the ratio of true positive to the sum of true positive and false positive from the confusion matrix as given by,

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Having a high precision indicates that out of the total alarms produced, the models were able to capture an increased number of algal bloom events.

Recall

End users frequently want the models to detect as many target events as possible correctly, as the cost of missing an event is high in certain domains. Recall measures the proportion of positive events captured by the model. It denotes the ratio of the number of true positive events captured by the model to the total number of positive events. Equation 5.3 presents the formula to calculate the recall score of a model,

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Thus, having a higher recall, it concludes that the models were able to capture a majority of the algal bloom events.

F-score

When evaluating a classification task, there is a trade-off between precision and recall since, at times, both precision and recall cannot alone assess the predictive ability of a model. As we want good scores of precision and recall, in our thesis, we use F-score, which balances between precision and recall. Since, we want to capture as many algal blooms (recall) and make sure the captured event is an algal bloom (precision), we use F-score to provide a balance. The formula is given by,

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.4}$$

5.1.2 Estimation Methods

As discussed in Section 2.3, the two primary estimation methods for time dependent data are Out-of-Sample (OOS) and Cross-Validation (CV). To recapitulate, in OOS methodology, a certain proportion of data is used to train the model and a subsequent part of the data is used for testing. In k-fold cross-validation (k-fold CV), the dataset

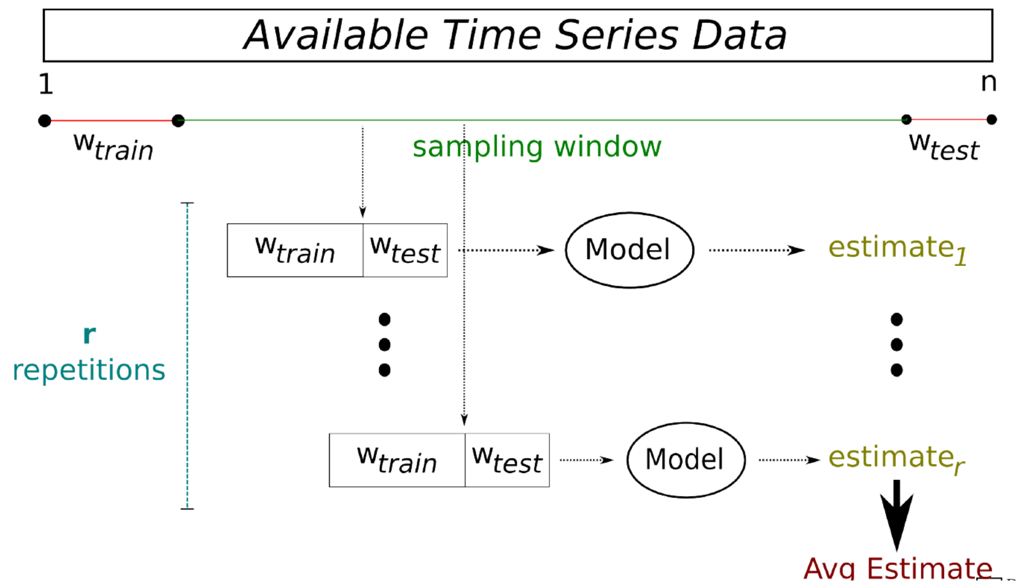


Figure 5.2: Monte Carlo simulation (figure taken from [73])

is partitioned into k-folds random samples. The method uses $k-1$ folds for training, and the remaining one fold is allocated for testing. This is repeated K times leaving each of the k folds as test set in turn, with the loss being evaluated in each iteration.

Cross-validation is not recommended in time series models due to the random shuffling of the data, thereby losing sequential timestamp information [19]. Since we have a time series dataset, we should never test on past data. Thus, we use Monte Carlo simulations [73] to evaluate our forecasting model. We randomly select a series of dates using the data before each of these dates for training and the subsequent data for testing, always preserving the sequential timestamps of the data. Many such iterations of Monte Carlo simulations are carried out with different train and test data sets, and the overall estimated of the performance is obtained by averaging over the scores achieved on each iteration. This process is depicted in Figure 5.2, where we have r repetitions with w_{train} and w_{test} representing the train and test window samples respectively. In each iteration, the model is trained on w_{train} samples and tested on the respective w_{test} samples, and the error estimate is calculated at each step. Finally, the average error estimate is calculated for the overall simulation over r repetitions.

We use the R package $performanceEstimation$ [72] to carry out these Monte Carlo experiments. This package contains a function named $performanceEstimation()$, that we use to run the experiments. In terms of parameterrs of this function we use the method *EstimationTask* to specify the metrics that are to be estimated, using the *metrics* parameter. Specifically, we set this parameter to the values F, *rec*, *prec* to represent F-score, Recall and Precision, respectively. Moreover, we indicate as estimation method the value *MonteCarlo*. In our experimental evaluation, we obtain the estimates using 20 repetitions of the Monte Carlo procedure with w_{train} of 30% and w_{test} of 30% of the available data.

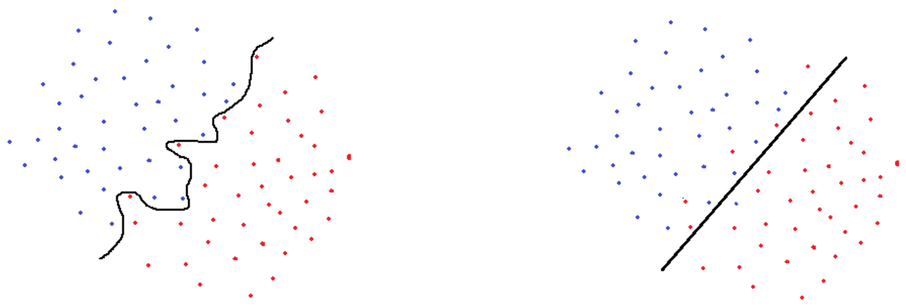
5.1.3 Forecasting models

In this section, we discuss the strategy used to select the parameter settings of the models which we apply to our data set. Hyperparameter tuning involves trying different values of the model's parameters to find an optimal setting that improves the models' performance. The experiments are carried out with different models using the Monte Carlo estimation method with 20 repetitions, each with 30% train and 30% test datasets. The models are evaluated on the metrics F-score, Precision, and Recall. The parameters are selected from many alternatives based on the impact it caused on the results of the algal bloom's forecasting task.

Parameter Tuning in Support Vector Machines (SVM)

To reiterate from the previous chapter, a Support Vector Machine (SVM) is a supervised machine learning algorithm used in classification and regression tasks. The main objective of SVM is to find the best hyperplane that classifies the data points. Thus, choosing the best hyperplane involves finding the right parameters for the model. We used the parameters *regularization* and *gamma* in our experiments, which influenced the most on the results to forecast the algal bloom events. We use function *svm* from R package $e1071$ [49] to model our forecasting task using SVM. We will discuss in detail each parameter in the following subsections.

Regularization (cost). If an SVM tries to correctly classify all the data points, it would be too specific in finding the decision boundary and would lead to overfitting [27]. This is called a hard-margin decision boundary, as shown in Figure 5.3 (a). It achieves good accuracy on training data but does not generalise well to the test



(a) Hard Margin SVM (b) Soft Margin SVM

Figure 5.3: Hard Margin and Soft Margin SVM
(figure taken from [82])

data. To overcome this issue, SVM uses a soft-margin decision boundary that allows for a few misclassifications but provides a significant margin between the support vectors and the decision boundary. Figure 5.3 (b) depicts a soft-margin where the decision boundary represents a line with some data being misclassified. The model with a soft-margin decision boundary accounts for a more generalized one rather than a model with hard-margin decision boundary.

Soft margin SVM concentrates on the following goals:

- to maximize the distance between decision boundary and support vectors
- to maximize the number of data points correctly classified

Some trade-offs exist between the goals mentioned above to obtain a decision boundary with reduced margin and all training data being correctly classified or having a decision boundary with high margins and some misclassified data points. This trade-off is controlled by the regularization parameter (cost) [34]. The regularization parameter (cost) penalizes for every misclassification. For high values of the cost parameter, the misclassification cost is high, resulting in a hard margin decision boundary. It tries to avoid any data point being misclassified and reduces the distance between data points and the decision boundary. Alternatively, if the cost of misclassification is smaller, it produces a model with a soft margin decision boundary with more misclassified data points. Thus, the regularization parameter influences the performance of the model and it is required to tune it to achieve best results.

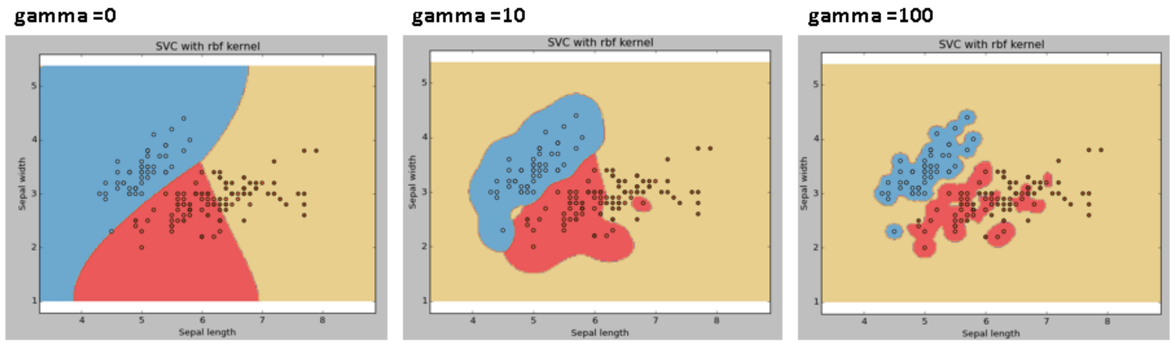


Figure 5.4: SVM with different gamma parameter
(figure taken from [60])

Gamma. Another parameter that influences the SVM model with Radial Basis Function (RBF) kernel is *gamma*, which represents the influence of a single data point on the model. Gamma controls the distance to which each data point influences the model [44]. Having low gamma values indicates the data points at far distances highly influence the model, and a high value of gamma indicates the influence of data points closer to the decision boundary. In other words, low values of gamma highly influence large groups with similar data points, and a high value means that the points have to be close enough to be considered as a group. Figure 5.4 explains different gamma values where a low value of 0 resulting in a large group, and any data point in blue or red can be classified to their respective group with loosely coupled boundary areas. However, with a gamma value of 100, groups data in a tightly bounded area of red or blue and any small noise could affect a data point to fall apart from the class boundary.

Finding the best parameter values in SVM. We experimented with different combinations of cost and gamma values on the train dataset with the experimental settings discussed in the above sections. We tried all combinations of cost values of 1, 5, 10, 100 and gamma values of 0.01, 0.05, 0.1, 0.5. Thus, we experimented with 16 different combinations, and the results are compared using the metrics discussed above. We achieved the best results on F-score with the combination of *cost* value of 1 and *gamma* value of 0.01 using the *svm* function from the R package e1071.

Random Forest

As we have discussed random forests in the previous chapter, we will discuss the hyperparameters and their importance in tuning them to improve the model's performance. As Random Forest consists of many different decision trees, the major hyperparameters that impact our results include the number of trees in the model and the number of features from which the best split test is selected at each decision node. We use the R package *randomForest* [45] to build the random forest model to forecast algal blooms. We experimented on the following parameters which had more impact on the results.

Number of trees. As Random Forest involves building many decision trees, the number of trees involved in the model is a crucial parameter. The variance is reduced in the model for a high number of trees, but the training time complexity increases. Thus, we tune the number of trees to achieve better performance. It is represented by the parameter *ntree* in *randomForest* method.

Minimum Sample Size on Leaf Nodes. It is essential for a random forest to manage the sample size on the nodes to consider them as leaves. It specifies the minimum number of samples to be present in the nodes to consider itself as a leaf [54]. Having a high minimum sample size makes the trees shrink and limits itself with smaller trees. It reduces the computation time for predictions but with some misclassified data. Figure 5.5 denotes a minimum sample value of 5. The leaf nodes that satisfy the constraint of a minimum sample size of 5 are denoted in green color. The node consisting of three samples do not meet the condition is represented in red color and is not considered a leaf. Thus, its parent node becomes the new leaf. The image on the right shows the final tree with new leaf nodes, which satisfy the constraint of a minimum sample size of 5. This parameter controls the growth of the tree by setting a minimum sample size on the leaf nodes. Low values of this parameter would make trees with larger depths. It is denoted by *nodesize* in the *randomForest* method, adjusted to improve the model's performance.

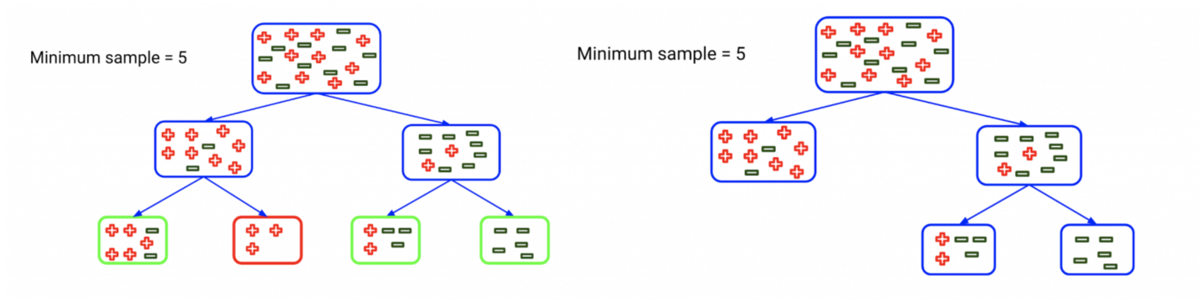


Figure 5.5: Minimum sample at leaf nodes (figure taken from [63])

Number of Candidate Features. The Random Forest model is highly influenced by the candidate features at each splitting node [54]. It denotes the number of candidate features that are randomly selected, from which a specific feature is chosen at each split when the tree is grown. Low values of this parameter result in more different and less correlated trees, which provides better stability when aggregating results. This parameter is denoted as $mtry$ in the $randomForest$ R package.

Finding the best parameter values in Random Forest. We experimented on the train dataset with different combinations of the number of trees with 500, 1000, 1500, 2000, minimum sample size on leaf nodes with 1, 2, 5, and the number of candidate features with 4, 5, 6 and $\sqrt{p}$, where p denotes the number of variables. We achieved the best results on F-score with the combination of $ntree$ as 1500, $nodesize$ with the default value of 1, and the number of candidate features ($mtry$) with $\sqrt{p}$ using the experimental settings discussed in the previous sections.

XGBoost

As discussed in previous chapter, XGBoost is a tree boosting algorithm commonly used in classification and regression tasks. In this section, we will be dealing with a few parameters used in our experiments which had more impact on the results. We use the R package $xgboost$ [26] to build our XGBoost model using the experimental settings discussed in the previous sections.

Number of Rounds. XGBoost undergoes numerous rounds of computation, represented by the number of rounds parameter. Every time the algorithm runs, it enhances the model by correcting the errors in the previous iterations. However, having a high number of rounds increases the computation time and effort. It is required to test the model with a combination of different number of rounds and other parameters to find the optimal value that maximizes performance. It is denoted by *nrounds* in the R package *xgboost*.

Learning Rate (ETA). As discussed earlier, XGBoost undergoes numerous rounds of computations and the weights are modified in each round. It also denotes the amount of correction at each step. Having a low *eta* will produce a more robust model since it makes very less correction at each step and reaches the best optimum with higher number of rounds. Having a low *eta* takes more rounds to train the model and thus, takes more computation time. However, having a high *eta* will produce results quicker at the cost of misclassifications, since it do not reach the best optimum within fewer number of rounds. It is represented by *eta* in the R function *xgboost* and the value ranges between 0 to 1, where an *eta* closer to 0 takes more time to process but produces better results.

Objective Function. The main task of an objective function in XGBoost is to reduce the loss in the model. We use $multi : softprob$ as the objective function that predicts the data point's probabilities belonging to each class. It is also required to specify the number of classes of the target variable in the dataset. We provide the number of classes in the target feature using *num_class* parameter as two since we build a two-class model to predict if the event is a bloom or normal.

Finding the best parameter values in XGBoost. We experimented over a different combination of the number of rounds with 500, 1000, 2000, learning rate over 0.1, 0.3, 0.5, 0.7 and with the objective function of $multi : softprob$ on the train dataset. We obtained best results on F-score with the combination of the number of rounds with 1000 and learning rate with 0.3 using the experimental settings discussed in the previous sections.

5.1.4 Class Imbalance

To reiterate from the previous chapter, class imbalance problem affects the model's performance on our forecasting algae blooms task, since we are interested in the rare events of algal blooms. Thus, we solve it by sampling the dataset using different methods discussed in Chapter 4. In this section, we will discuss the parameters and tuning them to improve the performance of the model.

Random Under-Sampling

As random under-sampling removes some samples belonging to the majority classes randomly with respect to the number of samples in minority classes, it is one of the simplest methods to deal with the class imbalance problem [16]. We use *RandUnder-Classif* function from the R package UBL [16] to perform random under-sampling on the training dataset. This method takes in a parameter called *C.perc* which accepts the values "balance", "extreme", and percentages of sampling of the classes. The value "balance" produces a balanced number of samples in all classes by removing few majority samples, whereas "extreme" would inverse the classes' existing proportion, transforming most frequent to less frequent and vice versa [16]. It also accepts percentage values between 0 and 1, where 1 represents no sampling, and other values represents the sampling percentage. For example, if *C.perc* is given a value of 0.2, the class samples are reduced to 20% of their original size.

We experimented with different values of *C.perc* parameter and different values of models' parameters along with the monte carlo experimental settings discussed in previous sections. The different values of *C.perc* parameters we experimented includes "balance", "extreme", 0.1, 0.4, 0.6 on each of the forecasting models. We achieved the best results on metric F-score using random under sampling on the models with the values "balance" on SVM, "extreme" on Random Forest and "balance" on XGBoost using the experimental settings discussed in previous sections.

Random Over-Sampling

As discussed in the previous chapter, random over-sampling introduces replicas of the least represented class to the dataset. Balancing the proportion of the samples in all classes by creating new samples of the minority class increases the dataset's size and also affects the computational capacity of the model. In our experiments, we use the function $RandOverClassif$ from the R package UBL to perform random over-sampling. This function accepts a parameter *C.perc* with the values "balance", "extreme" and percentages of over-sampling. The value "balance" creates minority samples until it obtains a balanced proportion of classes and "extreme" reverses the proportion making the least represented to be the most and vice versa. It also takes in percentage of over-sampling with values greater than or equal to 1. For example, the number of minority samples is doubled if *C.perc* takes a value 2 [16].

We experimented with different values of *C.perc* parameter and different values of models' parameters with Monte Carlo experimental settings discussed in previous sections. The different values of *C.perc* parameter we experimented include "balance", "extreme", 1.5, 2 and 2.5 on the previously mentioned models. We achieved the best results on metric F-score using random over sampling on the models with the values "extreme" on SVM, "extreme" on Random Forest and "balance" on XGBoost using the experimental settings discussed in previous sections.

SMOTE

SMOTE (Synthetic Minority Over-sampling TEchnique) introduces minority samples by synthetic creation and also removes certain samples of the least important class. As we have explained the process of creating minority samples in the previous chapter, we will deal about parameter tuning to achieve best performance using SMOTE method. We use the function $SmoteClassif$ from the R package UBL that implements SMOTE sampling, which accepts a parameter *C.perc*. This parameter takes in the values "balance", "extreme", where "balance" provides a balanced proportion of classes and "extreme" reverses the proportion making the least represented to be the most and vice versa. This parameter also takes in percentages of over or/and under sampling of the classes representing over-sampling, if the value is greater than 1 and under-sampling, if the value is less than 1. The other parameters that had more impact on the results include k, representing the number of nearest neighbors that is used to generate the samples [16], and the *dist* parameter that provides the distance metric used in the nearest neighbors.

We experimented SMOTE sampling with different values of *C.perc*, k and *dist* along with the different parameter values of the forecasting models using Monte Carlo experimental settings discussed in previous sections. We experimented with the different values of *C.perc* with "balance", "extreme", 1.5, 2, 0.1 and 0.3, k with 4, 5, 6, and *dist* with "Euclidean" and "Manhattan". We achieved the best results on F-score using Smote sampling on the models with the values "balance" on SVM, "extreme" on Random Forest and "balance" on XGBoost for the parameter *C.perc* with k as 5 and *dist* as "Euclidean" along with other settings discussed in previous sections.

Overall, we achieved the best results with the parameters *cost* as 1 and *gamma* with 0.01 for SVM, *ntree* as 1500, *nodesize* with 1, and the number of candidate features with $\sqrt{p}$ for Random Forest and *nrounds* as 1000 and learning rate with 0.3 for XGBoost model, with Monte Carlo experiments of 20 repetitions each trained on 30% samples and tested on 30% samples when evaluated on F-score metric. On sampling the original dataset, we achieved the best results on the models SVM using Smote with *C.perc* as "balance", k as 5 and *distance* as "Euclidean", Random Forest using Random Under Sampling with *C.perc* as "extreme" and XGBoost using Random Under Sampling with *C.perc* as "balance" apart from the experimental settings discussed above.

5.2 Results and Discussion

This section deals with the discussion and comparison of the results. We carried out the experiments on two different datasets – sampled and non-sampled datasets. Although the overall motivation of our proposed method is to avoid having to measure algae in water samples, and simply using mussels to forecast these values, we wanted to see how far we were from the results we would obtain if we were allowed to use the algae counts in the previous timestamps as predictors used by the models. Thus, we experimented with predictor variables including algae counts at recent timestamps (t and $t-1$) and without the algae counts as predictors. In summary, we experimented on the combinations of sampled and non-sampled datasets with and without algae counts as predictors.

The machine learning models average and standard deviation results on the sampled dataset in terms of the metrics F-score, recall, and precision are shown in Tables 5.1 and 5.2. The best results across different metrics and time horizons are highlighted in bold. From Table 5.1, it can be found that XGBoost performed well across all forecasting horizons (30 minutes, 2 hours, and 4 hours) on the metrics F-score and precision with the best values at 0.62 ± 0.1 and 0.52 ± 0.1 respectively, over a 30-minute forecasting task. However, we achieved the best results using Random Forest with 0.80 ± 0.05 on recall on the 30-minute forecasting task. When evaluated on the 2-hour task, we achieved the best results using XGBoost on F-score and precision with 0.61 ± 0.1 and 0.51 ± 0.1 respectively and Random Forest on recall with 0.80 ± 0.05. With a 4-hour forecasting task, we achieved the best results using XGBoost model on F-score and precision with 0.61 ± 0.1 and 0.51 ± 0.1, respectively and using Random Forest on recall with a score of 0.80 ± 0.05. We conclude from the results that XGBoost performed well with the best results over all future time horizons on the metrics F-score and precision, and Random Forest achieved better results on recall on the sampled dataset without algae counts as predictors. Overall, Random Forest model produced a recall of 0.80, which means that it was able to forecast 80% of the algal blooms. XGBoost produced a precision of 0.52, which means that from the bloom alarms issued by the model only 52% were correct, the others being false alarms.

The results of the experiment using algae counts as predictors on the sampled dataset are illustrated in Table 5.2. From the results, we found that XGBoost performed well with values 0.98 ± 0.03, 0.99 ± 0.03, and 0.97 ± 0.04 on the metrics F-score, recall, and precision, respectively, on 30-minute task. It is also shown that XGBoost achieved the best results over 2-hour forecasting task with 0.94 ± 0.05, 0.99 ± 0.02 and 0.91 ± 0.04 and over 4-hour task with 0.90 ± 0.04, 0.97 ± 0.03 and 0.85 ± 0.03 respectively, followed by Random Forest and SVM.

It is clear from the results of the two Tables 5.1 and 5.2 that the sampled dataset with algae counts at t and $t-1$ timestamps as predictors produced better results than the sampled dataset without those timestamps' algae counts as predictors. Overall, from the above two experiments' results, XGBoost achieved the best performance, followed by Random Forest and SVM on the sampled datasets. Also, the results on the sampled dataset depict that the models provide better forecasts over shorter periods into the future compared to far in time, although the differences are small.

Tables 5.3 and 5.4 illustrates the results on the metrics F-score, recall, and precision on the original non-sampled dataset without algae counts as predictors and

Algorithm	30 mins			2 hours			4 hours		
	F	**Rec**	**Prec**	**F**	**Rec**	**Prec**	**F**	**Rec**	**Prec**
SVM	0.59 ±0.1	0.78 ±0.06	0.49 ±0.1	0.60 ±0.1	0.79 ±0.02	0.49 ±0.1	0.59 ±0.1	0.78 ±0.05	0.49 ±0.1
RF	0.61 ±0.1	**0.80 ±0.05**	0.49 ±0.1	0.60 ±0.1	**0.80 ±0.05**	0.48 ±0.1	0.60 ±0.1	**0.80 ±0.05**	0.49 ±0.1
XGBoost	**0.62 ±0.1**	0.79 ±0.07	**0.52 ±0.1**	**0.61 ±0.1**	0.78 ±0.04	**0.51 ±0.1**	**0.61 ±0.1**	0.78 ±0.04	**0.51 ±0.1**

Table 5.1: Results without algae count as predictors on sampled dataset

Algorithm	30 mins			2 hours			4 hours		
	F	**Rec**	**Prec**	**F**	**Rec**	**Prec**	**F**	**Rec**	**Prec**
SVM	0.86 ±.03	0.98 ±.02	0.78 ±.05	0.86 ±.03	0.98 ±.03	0.78 ±.04	0.86 ±.05	0.97 ±.04	0.77 ±.05
RF	0.97 ±.03	0.98 ±.04	0.94 ±.03	0.92 ±.03	0.98 ±.03	0.90 ±.03	0.89 ±.02	0.93 ±.02	0.83 ±.02
XGBoost	**0.98 ±.03**	**0.99 ±.03**	**0.97 ±.04**	**0.94 ±.05**	**0.99 ±.02**	**0.91 ±.04**	**0.90 ±.04**	**0.97 ±.03**	**0.85 ±.03**

Table 5.2: Results with algae count as predictors on sampled dataset

with algae counts at recent timestamps, respectively. The non-sampled datasets' results without the algae count information at previous timestamps are illustrated in Table 5.3. It depicts that XGBoost performed well with the value of 0.55 ± 0.1 on F-score and Random Forest achieved better results over recall and precision with 0.65 ± 0.2 and 0.54 ± 0.1 respectively, over the 30-minute forecasting task. With 2 hours forecasting task, Random Forest and XGBoost achieved similar results on F-score with 0.54 ± 0.1, and Random Forest performed better over recall and precision with 0.64 ± 0.2 and 0.54 ± 0.1, respectively. Over the 4-hour forecasting task, Random Forest and XGBoost achieved similar results on F-score and recall with 0.54 ± 0.1 and 0.63 ± 0.2, but Random Forest performed well on precision with 0.54 ± 0.1.

The results from Table 5.4 depict that XGBoost performed well on all metrics (F-score, recall, and precision) with values 0.94 ± 0.03, 0.95 ± 0.04 and 0.93 ± 0.05 respectively, over a 30-minute forecasting task, followed by Random Forest and SVM. When evaluated on a 2-hour forecasting task, XGBoost achieved better results on F-score, recall, and precision with 0.93 ± 0.03, 0.94 ± 0.05, and 0.93 ± 0.05 respectively, and on a 4-hour forecasting task, with F-score of 0.93 ± 0.05, recall of 0.94 ± 0.05 and precision of 0.93 ± 0.05. It is also found that the models performed well in forecasting algae blooms over 30 minutes compared to 2 hours and 4 hours time periods, but the differences are pretty small. From Tables 5.3 and 5.4, it clearly shows that the models performed well in forecasting algae blooms when algae counts in recent timestamps (previous and current) are used as predictors. Overall, on the original non-sampled dataset, XGBoost performed well with algae counts as predictors, and Random Forest performed well without algae counts as predictors.

Algorithm	**30 mins**			**2 hours**			**4 hours**		
	F	**Rec**	**Prec**	**F**	**Rec**	**Prec**	**F**	**Rec**	**Prec**
SVM	0.49 ±0.1	0.48 ±0.2	0.50 ±0.1	0.49 ±0.1	0.48 ±0.2	0.49 ±0.1	0.48 ±0.1	0.48 ±0.2	0.48 ±0.1
RF	0.54 ±0.1	**0.65 ±0.2**	**0.54 ±0.1**	**0.54 ±0.1**	**0.64 ±0.2**	**0.54 ±0.1**	**0.54 ±0.1**	**0.63 ±0.2**	**0.54 ±0.1**
XGBoost	**0.55 ±0.1**	0.64 ±0.2	0.53 ±0.1	**0.54 ±0.1**	0.63 ±0.2	0.53 ±0.1	**0.54 ±0.1**	**0.63 ±0.2**	0.52 ±0.1

Table 5.3: Results without algae count as predictors on non-sampled dataset

Algorithm	**30 mins**			**2 hours**			**4 hours**		
	F	**Rec**	**Prec**	**F**	**Rec**	**Prec**	**F**	**Rec**	**Prec**
SVM	0.88 ±.02	0.83 ±.03	0.80 ±.03	0.88 ±.03	0.82 ±.04	0.80 ±.04	0.87 ±.02	0.82 ±.04	0.79 ±.05
RF	0.88 ±.02	0.84 ±.04	0.80 ±.03	0.88 ±.04	0.82 ±.04	0.79 ±.05	0.88 ±.02	0.82 ±.05	0.79 ±.04
XGBoost	**0.94 ±.03**	**0.95 ±.04**	**0.93 ±.05**	**0.93 ±.03**	**0.94 ±.05**	**0.93 ±.04**	**0.93 ±.05**	**0.94 ±.05**	**0.93 ±.05**

Table 5.4: Results with algae count as predictors on non-sampled dataset

It is evident from the results of the four experiments illustrated in the above tables that the models achieved better results on the sampled dataset compared to original non-sampled dataset. It is also shown that tuning certain parameters of the forecasting models and sampling methods produces better results. From the experiments, we found that Random Forest performed well on the non-sampled dataset without algae counts as predictors. However, XGBoost achieved better results in all other experiments evaluated on the metrics F-score, recall, and precision. The models were able to forecast about 80% of the algal blooms and out of the total alarms issued, the models were able to forecast about 52% algal blooms. The results also depict that the models are better in detecting the algal bloom events within a shorter time interval into the future (30 minutes) compared to 4 hours. However, the predictions at time intervals of 2 and 4 hours are not so bad and are just behind the forecasts of

30-minute time intervals. As domain experts are interested in forecasting long periods into the future, it depends on the domain experts to critically decide the specific time horizon. Finally, it is shown that knowing the algae counts at the current and previous timestamps makes it more convenient for the models to predict future algal bloom events. As discussed in the previous chapter, our main objective is to detect algae blooms without measuring the algae counts at current timestamps. Thus, we focus on the experiments without the algae counts as predictors since it becomes challenging to measure the algae count at every timestamp, which violates our main objective.

5.2.1 Statistical Significance of the Reported Results

It is essential to test whether the results we have reported are statistically significant, i.e. to refute the hypothesis that the results are obtained by chance. We compared the different approaches on a single task as well as on multiple tasks. We used Wilcoxon Signed Ranks test [78] to test the statistical significance of the differences between the approaches on a single task, and Friedman test [33] to compare them on multiple tasks, followed by Nemenyi post-hoc test [28]. Since we are mainly interested on the sampled dataset without algae counts as predictors, we focused this analysis of the statistical significance on this dataset involving the forecasting tasks with different future time horizons: 30 minutes, 2 hours, and 4 hours. The approach towards the predictive task is known as workflow. It defines the implementation of the predictive task given the train and test datasets. We define different workflows of our forecasting task using the models SVM, Random Forest and XGBoost, where we perform the implementation of the predictive task.

Comparison of Two Classifiers — Wilcoxon Signed Ranks Test

Wilcoxon Signed Ranks Test ranks the differences in the performance among two classifiers on a task [28]. The null hypothesis of this test states that the difference between the two workflows is zero. That means the hypothesis is rejected if the result of the test has a p-value less than a specific threshold (typically 0.01 or 0.05, i.e., 99% or 95% confidence to reject the hypothesis). We chose p-value as 0.05 in our experiments on the Wilcoxon Signed Rank test since we wanted to achieve a minimum of 95% confidence, i.e., the hypothesis is rejected if $p - value < 0.05$. The results of the Wilcoxon Signed Ranks Test against each metric(F-score, recall, and precision) on all tasks of different time horizons(30 minutes, 2 hours, and 4 hours) are provided below,

- F-score: The results of the tests on all models have $p - value > 0.05$ and do not reject the null hypothesis on all tasks when evaluated on F-score.
- Recall: The results of the test on recall depict that it has significant differences when compared among XGBoost and Random Forest and thus rejects the null hypothesis. Whereas the results depict that XGBoost and SVM, SVM and Random Forest are not significantly different, accepting the null hypothesis on all tasks.
- Precision: The results of the test on precision show that it rejects the null hypothesis and has significant differences among SVM and XGBoost on all tasks but accept the hypothesis among SVM and Random Forest, Random Forest and XGBoost on all forecasting tasks.

The above results are also summarized in table below,

Algorithm	**F-score**			**Recall**			**Precision**		
	SVM	**RF**	**XGBoost**	**SVM**	**RF**	**XGBoost**	**SVM**	**RF**	**XGBoost**
SVM	NA	Accept	Accept	NA	Accept	Accept	NA	Accept	**Reject**
RF	Accept	NA	Accept	Accept	NA	**Reject**	Accept	NA	Accept
XGBoost	Accept	Accept	NA	Accept	**Reject**	NA	**Reject**	Accept	NA

Table 5.5: Wilcoxon Signed Ranks Test on sampled dataset without algae counts as predictors over all time horizons

Comparison of Multiple Classifiers — Friedman Test and Nemenyi Post-Hoc Test

As Wilcoxon Signed Rank Test compares only two classifiers, Friedman test compares multiple algorithms and workflows. Friedman test's null hypothesis states that all workflows are equivalent and the rankings across all the tasks are also equal [33]. The results of the Friedman test on all the classifiers and workflows show that all the workflows reject the null hypothesis. Thus, we performed Nemenyi post-hoc test to perform paired comparisons among all pairs of workflows. The null hypothesis of Nemenyi test states that there is no significant differences among the ranks of certain pair of workflows [28]. We performed the Nemenyi test among the different algorithms on multiple tasks of the different time horizons (30 minutes, 2 hours, and 4 hours), and the results are depicted below,

- F-Score: Rejects null hypothesis among SVM and XGBoost and accepts among SVM and Random Forest, Random Forest and XGBoost.
- Recall: Rejects null hypothesis among SVM and Random Forest, but accepts among SVM and XGBoost, Random Forest and XGBoost.
- Precision: Rejects null hypothesis among SVM and XGBoost and accepts among SVM and Random Forest, Random Forest and XGBoost.

The results of Nemenyi test are summarized in the table below,

Algorithm	**F-score**			**Recall**			**Precision**		
	SVM	**RF**	**XGBoost**	**SVM**	**RF**	**XGBoost**	**SVM**	**RF**	**XGBoost**
SVM	NA	Accept	**Reject**	NA	**Reject**	Accept	NA	Accept	**Reject**
RF	Accept	NA	Accept	**Reject**	NA	Accept	Accept	NA	Accept
XGBoost	**Reject**	Accept	NA	Accept	Accept	NA	**Reject**	Accept	NA

Table 5.6: Nemenyi post-hoc Test on sampled dataset without algae counts as predictors over all time horizons

The above conducted tests depict the statistical significance to prove that the results are not obtained by chance. The results of Wilcoxon Signed Ranks test show that the differences among the workflows of Random Forest and XGBoost are statistically significant when evaluated for 95% confidence score on recall. Also, the differences among the workflows of SVM and XGBoost are statistically significant over precision, however the workflows are not statistically significant in their differences when evaluated over F-score for 95% confidence score. Nemenyi post-hoc test shows that the differences among the workflows on multiple tasks are statistically significant among SVM and XGBoost when evaluated on F-score and precision, whereas, the difference among the workflows of SVM and Random Forest are statistically significant when evaluated on recall.

In summary, the predictive models performed better on forecasting tasks involving shorter time intervals (30 minutes) compared to 4 hours, though the results were not bad on a 4-hour forecasting task. Our results also show that the models performed better on sampled datasets compared to the non-sampled (original) dataset. From the results, we conclude that the models were able to forecast 80% of the total algal blooms although with a considerable amount of false alarms (48%). From the statistical significance tests, we conclude that the results were not obtained by chance and with several observed differences being different with 95% statistical confidence.

Chapter 6

Conclusions

Time series data occurs in many application domains like finance, healthcare, energy, and several others. Forecasting and activity monitoring are two frequent tasks associated with this type of data. They support the application of data-driven decisions to some of the organizations' challenges allowing them to mitigate or prevent losses.

This thesis main objective involves activity monitoring that leverages time series data to anticipate algal blooms in aquaculture farms. The detection of algal blooms helps domain experts in making decisions to improve water quality. We defined approaches to build machine learning models in the context of forecasting algal blooms. We leveraged the historical data and experimented to predict algal bloom events in several future time horizons: 30 minutes, 2 hours, and 4 hours.

Following the advice of domain experts, our approach used as main source of information data concerning the valve openings of a series of mussels that were placed in the aquaculture farms. The first step of our approach consisted of a series of feature engineering steps designed to produce a set of predictors based on the valve openings that could help in the task of forecasting blooms. We have applied several machine learning algorithms, such as SVM, Random Forest, and XGBoost, to the resulting dataset. The models were learned with the goal of forecasting whether an algae bloom was to occur. We have solved this task for three different forecasting horizons. We have tried these approaches for different parameter settings, which were evaluated and compared in Chapter 5.

A few sampling methodologies to balance the distribution of the target variable in our our dataset have been tried since many machine learning models do not work well with an imbalanced dataset [17]. We used a few sampling methodologies such as Random Under Sampling, Random Over Sampling, and SMOTE sampling to balance our datasets. We have tried a few parameters of sampling methods and tuned them to achieve better results. Based on the results in Chapter 5, we found that these sampling strategies led to better results than the original imbalanced datasets.

We achieved the task of forecasting algal blooms with an F-score of 0.62, and precision of 0.52 using XGBoost, and a recall of 0.80 using a Random Forest model when evaluated on a 30-minute forecasting task without using algae counts as predictors at t and $t-1$ timestamps. This means the models were able to forecast 80% of the algal blooms at the cost of 48% false alarms. Our results also show that the scores on a 4-hour forecasting task are just behind 30-minute forecasting task with 0.61 and 0.51 for F-score and precision, respectively, and 0.80 for recall.

We also achieved the best scores on F-score, recall and precision with 0.98, 0.99 and 0.97, respectively using XGBoost model when algae counts at t and $t-1$ timestamps are included as predictors on a 30-minute forecasting task. Thus, the results depict that if the algae counts at the current, t and previous timestamps, $t-1$ are used as predictors, the models produced more precise and better results forecasting 99% of the total algal blooms at the cost of 3% false alarms. However, it involves measuring the current algae counts in the water at every timestamp, which is what we want to avoid by using readings from mussels instead. Overall, we achieved the best results with the sampled dataset compared to original dataset and on forecasting algal blooms for time periods with a shorter time interval into the future (30 minutes) rather than 4 hours, although the differences are small.

6.1 Future Work

The methods discussed in this thesis can be improved in a number of ways leading to the future work of this thesis. In this section, we outline some of the potential lines of work that could build upon our thesis to achieve more precise forecasts in the detection of algal blooms.

We have modeled the forecasting task to detect algal blooms using various machine learning models, which support domain experts in making decisions concerning the aquaculture farms. Domain experts suggest that other algae species also affect the micro closures in the mussels' valve openings. Due to the unavailability of the other algae species data, we limited ourselves to the species *Alexandrium Tamarense*. Thus, the predictions of algal blooms may be able to be improved with the integration of other algae species' counts and analyzing mussels' behavior on those species.

We modeled the forecasting task based on the four mussels' valve opening percentage. However, having valve opening information of more mussels produces more data for the models to learn the predictive task and could improve the predictions. In the future, data from more mussels could be used to improve the performance of the models in achieving better forecasts of algal blooms.

Bibliography

[1] Exponential smoothing. https://otexts.com/fpp2/expsmooth.html.

[2] Mean squared logarithmic error (msle). https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle).

[3] R2 score and explained variance. https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics.

[4] Simple exponential smoothing. https://otexts.com/fpp2/ses.html.

[5] Theil inequality coefficient. https://www.vosesoftware.com/riskwiki/Thielinequalitycoefficient.php.

[6] Theil inequality coefficient. https://www.statisticshowto.com/uncertainty-coefficient/.

[7] Rahul Agarwal. The 5 classification evaluation metrics every data scientist must know. https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226, 2019.

[8] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[9] John Asafu-Adjaye. The relationship between energy consumption, energy prices and economic growth: time series evidence from asian developing countries. *Energy economics*, 22(6):615–625, 2000.

[10] Luís Baía and Luís Torgo. A comparative study of approaches to forecast the correct trading actions. *Expert Systems*, 34(1):e12169, 2017.

[11] Luís Carlos Gouveia Baía. Actionable forecasting and activity monitoring: applications to financial trading. Master's thesis, University of Porto, 2015.

[12] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[13] Saptashwa Bhattacharyya. Support vector machine: Kernel trick; mercer's theorem. https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d, 2018.

[14] Charlotte Bourne. Forecasting with machine learning techniques. `https://www.cardinalpath.com/blog/forecasting-with-machine-learning-techniques`, 2016.

[15] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[16] Paula Branco, Rita P Ribeiro, and Luis Torgo. Ubl: R package for utility-based learning. *arXiv preprint arXiv:1604.08079*, 2016.

[17] Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):31, 2016.

[18] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] Vitor Cerqueira, Luis Torgo, and Igor Mozetič. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11):1997–2028, 2020.

[20] Vítor Manuel Araújo Cerqueira. Ensembles for time series forecasting. 2019.

[21] Afroz Chakure. An introduction to decision tree classifier. `https://medium.com/@aaaanchakure/decision-tree-classification-de64fc4d5aac`, 2019.

[22] Ngai Hang Chan. *Time series: applications to finance*, volume 487. John Wiley & Sons, 2004.

[23] Chris Chatfield. *Time-series forecasting.* CRC press, 2000.

[24] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[25] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[26] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, and Yutian Li. *xgboost: Extreme Gradient Boosting*, 2020. R package version 1.0.0.2.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[28] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[29] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3):291–316, 1997.

[30] Tom Fawcett and Foster Provost. Activity monitoring: Noticing interesting changes in behavior. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 53–62. ACM, 1999.

[31] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[32] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[33] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[34] Frauke Friedrichs and Christian Igel. Evolutionary tuning of multiple svm parameters. *Neurocomputing*, 64:107–117, 2005.

[35] Rohit Gandhi. Support vector machine — introduction to machine learning algorithms. `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`, 2018.

[36] Iman Ghalehkhondabi, Ehsan Ardjmand, Gary R Weckman, and William A Young. An overview of energy demand forecasting methods published in 2005–2015. *Energy Systems*, 8(2):411–447, 2017.

[37] Mohamed F Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. Extraction of interpretable multivariate patterns for early diagnostics. In *2013 IEEE 13th International Conference on Data Mining*, pages 201–210. IEEE, 2013.

[38] Shameek Ghosh, Mengling Feng, Hung Nguyen, and Jinyan Li. Hypotension risk prediction via sequential contrast patterns of icu blood pressure. *IEEE journal of biomedical and health informatics*, 20(5):1416–1426, 2015.

[39] Shameek Ghosh, Mengling Feng, Hung Nguyen, and Jinyan Li. Hypotension risk prediction via sequential contrast patterns of icu blood pressure. *IEEE journal of biomedical and health informatics*, 20(5):1416–1426, 2016.

[40] Benjamin Graham and Jason Zweig. *The intelligent investor*. HarperBusiness Essentials New York, USA, 2003.

[41] Adam Hayes. R-squared definition. `https://www.investopedia.com/terms/r/r-squared.asp`, 2020.

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

[43] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[44] Petre Lameski, Eftim Zdravevski, Riste Mingov, and Andrea Kulakov. Svm parameter tuning with grid search and its impact on reduction of model overfitting. In *Rough sets, fuzzy sets, data mining, and granular computing*, pages 464–474. Springer, 2015.

[45] Andy Liaw and Matthew Wiener. *Classification and Regression by randomForest*, 2018. R package version 4.6-14.

[46] Chi-Jie Lu, Tian-Shyug Lee, and Chih-Chou Chiu. Financial time series forecasting using independent component analysis and support vector regression. *Decision support systems*, 47(2):115–125, 2009.

[47] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889, 2018.

[48] Allan DR McQuarrie and Chih-Ling Tsai. *Regression and time series model selection.* World Scientific, 1998.

[49] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2019. R package version 1.7-3.

[50] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, 2013.

[51] Usue Mori, Alexander Mendiburu, Eamonn Keogh, and Jose A Lozano. Reliable early classification of time series based on discriminating the classes over time. *Data mining and knowledge discovery*, 31(1):233–263, 2017.

[52] Mussels. Mussels — Wikipedia, the free encyclopedia, 2020. [Online; accessed 10-September-2020].

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

[54] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301, 2019.

[55] Joseph Prusa, Taghi M Khoshgoftaar, David J Dittman, and Amri Napolitano. Using random undersampling to alleviate class imbalance on tweet sentiment data. In *2015 IEEE international conference on information reuse and integration*, pages 197–202. IEEE, 2015.

[56] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[57] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/iaai, Vol. 1*, pages 725–730, 1996.

[58] Jeff Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of econometrics*, 99(1):39–61, 2000.

[59] Ed Ramsden. Forecasting - metrics for time series forecasts. `http://www.edscave.com/forecasting---time-series-metrics.html`, 2016.

[60] Sunil Ray. Understanding support vector machine(svm) algorithm with examples. `https://analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/`, 2017.

[61] Alaa Sagheer and Mostafa Kotb. Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing*, 323:203–213, 2019.

[62] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[63] Sharoon Saxena. Random forest hyperparameter tuning. `https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/`, 2020.

[64] Alakh Sethi. Support vector regression tutorial for machine learning. `https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/`, 2020.

[65] Md Sumon Shahriar, Ashfaqur Rahman, and John McCulloch. Predicting shellfish farm closures using time series classification for aquaculture decision support. *Computers and electronics in agriculture*, 102:85–97, 2014.

[66] Tom Sharp. An introduction to support vector regression (svr). `https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2`, 2020.

[67] Sima Siami-Namini and Akbar Siami Namin. Forecasting economics and financial time series: Arima vs. lstm. *arXiv preprint arXiv:1803.06386*, 2018.

[68] Tom AB Snijders. On cross-validation for predictor evaluation in time series. In *On Model Uncertainty and its Statistical Implications*, pages 56–69. Springer, 1988.

[69] Peter Stone and Manuela Veloso. Layered learning. In *European Conference on Machine Learning*, pages 369–381. Springer, 2000.

[70] Kanchan M Tarwani and Swathi Edem. Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, 48:301–304, 2017.

[71] Leonard J Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. *International journal of forecasting*, 16(4):437–450, 2000.

[72] L. Torgo. An infra-structure for performance estimation and experimental comparison of predictive models in r. *CoRR*, abs/1412.0436, 2014.

[73] Luis Torgo. Lecture notes in performance estimation, November 2019.

[74] Luis Torgo. Lecture notes in support vector machines, October 2019.

[75] Ruimei Wang, Daqing Chen, and Zetian Fu. Awqee-dss: A decision support system for aquaculture water quality evaluation and early-warning. In *2006 International Conference on Computational Intelligence and Security*, volume 2, pages 959–962. IEEE, 2006.

[76] Andreas S Weigend. *Time series prediction: forecasting the future and understanding the past.* Routledge, 2018.

[77] Gary M Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *KDD*, pages 359–363, 1998.

[78] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[79] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[80] Kenji Yamanishi and Jun-ichi Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 676–681. ACM, 2002.

[81] Tony Yiu. Understanding random forest. `https://towardsdatascience.com/understanding-random-forest-58381e0602d2`, 2019.

[82] Soner Yıldırım. Hyperparameter tuning for support vector machines — c and gamma parameters. `https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167`, May 2020.

[83] Fan Zhang, Chirag Deb, Siew Eang Lee, Junjing Yang, and Kwok Wei Shah. Time series forecasting for building energy consumption using weighted support vector regression with differential evolution optimization technique. *Energy and Buildings*, 126:94–103, 2016.