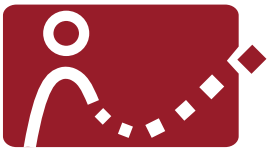
Data-Driven Fault Detection and Isolation of Industrial Fluid Transfer Systems



Masterarbeit von Debanjan Guha Neogi
Tag der Einreichung: 2. Februar 2018

1. Gutachter: Prof. Dr. Max Mühlhäuser
2. Gutachter: Sebastian Kauschke

Darmstadt, 2. Februar 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Telekooperation
Prof. Dr. Max Mühlhäuser

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Name: Debanjan Guha Neogi Datum: Unterschrift:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Name: Debanjan Guha Neogi Date: Signature:

Abstract

One of the key promises of Industry 4.0 is to reshape the various industrial sectors through the integration of advanced analytics and IoT into its day to day operations. A major component of Industry 4.0 is predictive maintenance, which allows real-time remote condition monitoring of critical industrial equipment, leading to reduced downtimes and optimal asset management. However, the application of analytical methods into existing industrial ecosystems is largely a non-standard problem and varies from case to case. In this thesis we provide an end-to-end solution for fault detection and isolation (FDI) in an industrial fluid transfer system, with the primary objective of finding leaks in the reservoirs using advanced signal processing and machine learning techniques. Optimization remains a driving force throughout the implementation, keeping in mind the environmental constraints and the limited hardware and sensing capabilities of the devices involved. We extract the relevant features from the available data-set and use it to train the predictive models. The algorithms consist of a single strong learner and ensemble of weak learners to provide comparative analysis for determining the best possible option. We employ Bayesian statistics to further improve the predictive outcomes. For the next part, we identify the optimal parameters for our use case through various experiments involving the different phases of our implementation life cycle. Finally, we design a prototype of our solution in the industrial setup to demonstrate the feasibility of the design concept. Our results show a prediction accuracy of over 99% for the reservoirs, while at the same time being fast, lightweight and unobtrusive to the existing test-bed. On top of that, our overall implementation and optimization techniques provide flexibility in terms of architectural and hardware changes, making it robust and future-proof.

Zusammenfassung

Eines der wichtigsten Versprechen von Industrie 4.0 ist die Neugestaltung der verschiedenen Industriesektoren durch die Integration von Advanced Analytics und IoT in den täglichen Betrieb. Ein wichtiger Bestandteil von Industrie 4.0 ist die vorausschauende Wartung, die eine Fernüberwachung von kritischen Industrieanlagen in Echtzeit ermöglicht. Dies führt zu geringeren Ausfallzeiten und einem optimalen Anlagenmanagement. Die Anwendung analytischer Methoden in bestehenden industriellen Ökosystemen ist jedoch weitgehend ein nicht standardisiertes Problem und variiert von Fall zu Fall. In dieser Arbeit bieten wir eine Ende-zu-Ende-Lösung für die Fehlererkennung und -isolierung (FDI) in einem industriellen Flüssigkeitstransfersystem mit dem Hauptziel, Lecks in den Reservoirs mittels fortschrittlicher Signalverarbeitung und Machinelearning zu finden. Die Optimierung bleibt eine treibende Kraft während der gesamten Implementierung, wobei die Umgebungsbedingungen und die begrenzten Hardware- und Sensorfähigkeiten der eingesetzten Geräte berücksichtigt werden. Wir extrahieren die relevanten Merkmale aus dem verfügbaren Datensatz und verwenden sie, um die Vorhersagemodelle zu verbessern. Die Algorithmen bestehen aus einem einzelnen starken Learner und einem Ensemble von schwachen Learner, die eine vergleichende Analyse zur Bestimmung der bestmöglichen Option liefern. Wir verwenden Bayes'sche Statistiken, um die Vorhersageergebnisse weiter zu verbessern. Für den nächsten Teil identifizieren wir die optimalen Parameter für unseren Anwendungsfall durch verschiedene Experimente, die die verschiedenen Phasen unseres Implementierungslebenszyklus betreffen. Schließlich entwerfen wir einen Prototyp unserer Lösung in industriellem Umfeld, um die Machbarkeit des Entwurfskonzepts zu demonstrieren. Unsere Ergebnisse zeigen eine Vorhersagegenauigkeit von über 99% für die Reservoirs, während sie gleichzeitig schnell, leicht und unauffällig auf dem vorhandenen Testbett sind. Darüber hinaus bieten unsere gesamten Implementierungs- und Optimierungstechniken Flexibilität in Bezug auf Architektur- und Hardwareänderungen und sind damit robust und zukunftssicher.

Contents

Acknowledgements

There are many people I wish to thank. To begin with I would like to express my sincere gratitude to my thesis advisor Sebastian Kauschke for all his help and guidance with the research and writing of the thesis. I am also thankful to Prof. Dr. Max Mühlhäuser and the Telecooperation Group in Technische Universität Darmstadt for having me as one of their thesis candidates. I would also like to acknowledge Dr. Alireza Alghassi and tiefNetz Technologies for giving me the opportunity to work on one of their key projects. I thank all the authors in the references that provided me with the groundwork for the research, with special mention to the ScikitLearn online community.

And finally, I would like to thank my parents for their support and encouragement throughout my years of study, and my life in general.

1 Introduction

Advancements in technology over the past decade have changed the outlook on how the various industries operate. Internet of things (IoT), cloud computing, big data and cognitive computing are gradually becoming an integral part of manufacturing and engineering processes, resulting in a whole new approach known as Industry 4.0 [24]. The term was first coined by the German government in one of its memos issued earlier in this decade, which proposed an outline to completely automate the manufacturing industry. The key concept of this approach is 'smart factories' which brings in higher levels of automation and self-optimization never conceived before, resulting in faster product roll-out and more streamlined resource management. Industry 4.0 is still in its early phases, and with increased investment both in terms of capital and technology, is poised to have long-standing effects on business processes and organizations.

Predictive maintenance is an integral part of this new system. Not so long ago maintenance of industrial equipment was more of a reactive approach, with periodic checks and troubleshooting that needed human intervention. However, the new industry paradigm proposes a fully automated proactive approach to this, known as condition monitoring. This consists of a wide variety of sensors and actuators collecting continuous data about the health of the equipment for a timely diagnosis to reduce outages and production downtime. Almost every condition monitoring system has machine learning and artificial intelligence as its underlying logic. However, the application of these methodologies is not standardized and differs from one use case to another. In this thesis, we look into one of such use cases. We develop a fault detection application for an industrial fluid transfer system using advanced signal processing and machine learning techniques. Our application is built on two confining determinants: it should adapt to the existing setup in an unobtrusive manner, and should be flexible enough to accommodate the changes in hardware. This makes resource usage and optimization a key deliverable for our implementation, along with the usual factors like prediction accuracy and fast response time. The former is achieved through a series of experiments combining the different parameters, while for the latter we have used probabilistic models to improve the predictive outcomes. A working prototype of our methodology is used on the existing setup as a proof of concept for our applied research. To summarize, we have realized a practical solution for condition monitoring to a specific unsolved problem with real business implications, and can be extended to similar problems with minor alterations in the implementation logic. The rest of the thesis is organized as follows:

In Chapter 2 we introduce some concepts of machine learning relevant to our applied research. This includes a brief explanation of the working principles of individual classifiers like support vector machines and decision trees, and also ensemble learners like randomized decision forests and gradient boosting methods. We state some common methods of parameter optimization for the models, and the commonly used metrics for evaluating the performance and accuracy of binary classification. We also introduce the concept of imbalanced class, which is one of the key properties of our data-set. For a binary classification, the problem of class imbalance arises when one class representation is significantly higher than the other. Finally we look into the concept of Bayesian Inference, that uses past results and statistical modelling to determine the probability of certain occurrences.

In Chapter 3 we briefly discuss about acoustic signals and their emission properties representative of our data. This includes the data collection methodology and some fundamental analysis to gain insight into the collected data. We also briefly mention the experimental setup of the data collection environment, which consists of various sensors and data acquisition systems.

Next, in Chapter 4 we apply feature engineering on our collected data to obtain what we consider meaningful features for our data-set. We segment the input data into various batches and compute the features for each batch. The features are essentially the time domain statistics for the input signal. We then proceed to understand the extracted features through univariate and multivariate analysis. For the next part we take care of the imbalanced class problem by down-sampling the majority class to be roughly equal to the other class. In the end we suggest a signal quantization technique to reduce the bitrate of the input signal, and the motivation behind this.

In Chapter 5 we present the results of our training models in a comparative manner. The comparison is done in terms of the accuracy metrics introduced in Chapter 2, and goes on to show the improvement each model has to offer over a pre-defined baseline. The baseline is defined to be a naive classification model, akin to random guessing.

In Chapter 6 we apply the concept of Bayesian Inference proposed in Chapter 2 to our data-set. The method uses the Bayes' theorem to update the probability of a certain occurrence with increasing arrival of input data. We design a custom Bayesian model specific to our use case, and present the results through various examples involving both hypothetical and real prediction data from our experiment. The results, when collated over to represent the prediction accuracy of a reservoir, shows the accuracy to be above 99%. We conclude with some error checking mechanism to verify the performance of our Bayesian model applied in the previous section.

Subsequently, in Chapter 7 we lay down the optimization techniques to select the best parameters for our use case. To begin with, we determine the minimum Bayesian window required to obtain the desired accuracy of over 99%. Next we determine the resource consumption of our various models and reach at a combination we believe is optimal for our use case.

Finally, in Chapter 8 we form a prototype as a proof of concept for our design principles. This includes implementation and adaptation of our applied research into the existing industrial setup. We briefly describe the pipeline for our implementation procedure including the gateway which is central to our design. In the end we create an online dashboard that provides a visual summary of our prediction results. To conclude, in Chapter 9, we present some other work related to our use case, including future work for the refinement of our design methodology.

2 Literature Review

This chapter consists of a detailed introduction to the various theoretical concepts and methodologies of machine learning that are relevant to our research.

2.1 Machine Learning - An Introduction

In the recent years Machine Learning (ML) seem to have caught the attention of computer scientists, mathematicians and engineers alike. However, it is not nearly a new or a radical concept, but has been prevalent for decades. It had its humble yet notable beginning in the 50s (Samuel's checker playing program) [45], but truly became mainstream as a spam filter in the 90s. Since then hundreds of applications have employed machine learning concepts one way or the other. Yet, it is only recently that we have begun to use ML to extremely large and complex systems with very high accuracy. Machine learning is now ubiquitous to the industrial and scientific world, and will only increase with improved hardware (GPUs) and services (Cloud, Big Data) being available at cheaper prices.

A Machine Learning algorithm differs from a traditional computer program. In the latter a program with certain input is run on the computer to obtain the desired output, while in machine learning we train the input data with an output to create a model that can then be used as a normal program afterwards (Figure 2.1). This can be seen as a way to automate the programming process, and let the computer choose and build a program that works best for a given set of problems. It has various advantages over traditional computing, like learning about the important features for a given task without any human intervention. On the other hand non ML approaches require handpicking useful attributes, which quickly becomes a problem with increasing data and is nearly impossible with very large data-sets. Also some problems are too complex for traditional methods or have no known algorithm, which can be easily handled with basic machine learning approaches.

Machine Learning systems can be broadly classified into certain categories based on the following assumptions.

- The level of human supervision involved in the training of ML models.
- Online vs batch learning
- Instance-based vs model-based

For the first part we can classify the ML systems based on the level of supervision (usually a reference target output) it received at the time of training. Based on this we can divide the types of ML into the following sub-categories [18].

Supervised Learning: The training data comes with the desired solutions, known as labels. Supervised learning tasks can be further divided into classification and regression. Classification deals with identifying class labels or group memberships for the data. A typical example can

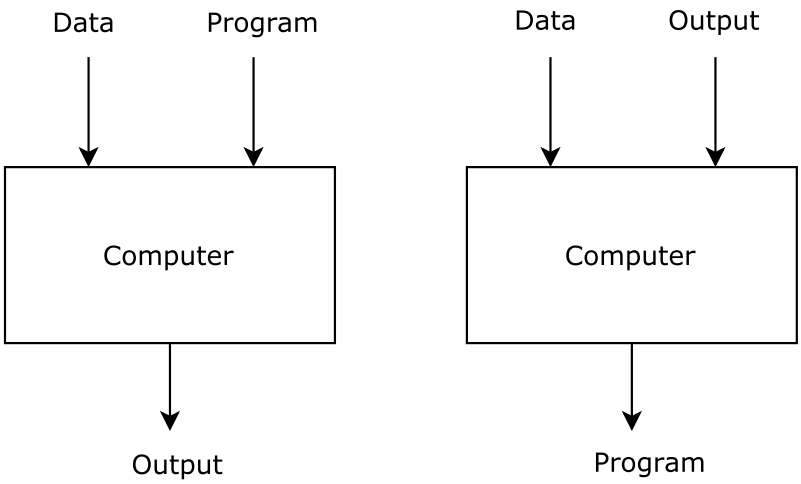
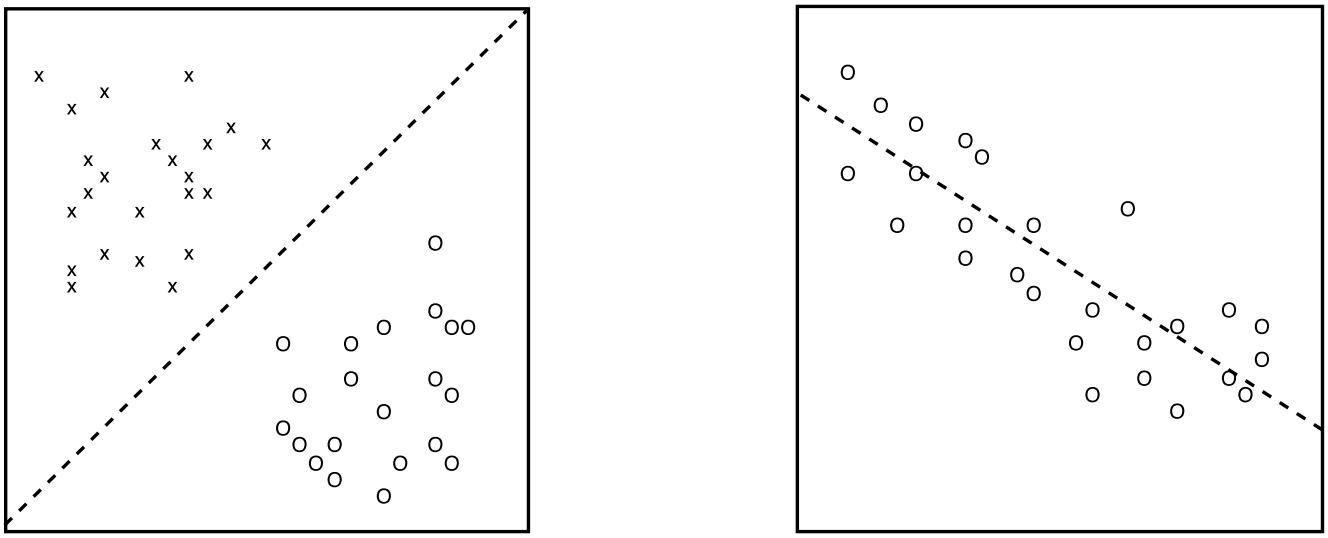


Figure 2.1.: A block diagram showing the difference between the fundamental concept of machine learning (right) with respect to traditional computing (left).

be a spam filter, where we consider only two distinct class labels i.e. whether an email is a spam or not. A common practice is to assign numeric values to distinct class labels like 0 for not spam and 1 for spam for the training data-set, and then use the trained model to classify new emails. Alternatively, we sometimes need to predict a continuous numeric value based on a given set of features. This type of learning is called regression. A typical example can be the prediction of house prices given some set of features like location, room sizes etc. Here the labels as well as the prediction output will not be distinct classes but a collection of continuous values. In other words while regression estimates a value from a continuous data-set, classification predicts the membership to a class (Figure 2.2). However, in some cases both can be used interchangeably by creating a categorical range for continuous predictive values (logistic regression). Although there is a wide range of supervised learning algorithms, we will only deal with some of the classification algorithms in the coming chapters to strictly adhere to the scope of the thesis.

Unsupervised Learning: The training data does not contain any labels or desired outputs. Depending on the underlying algorithm, its primary task is to recognize patterns or structural relations in the data-set and group them into various clusters based on those patterns. Some common methods for unsupervised learning include hierarchical and K-means clustering, hidden Markov models, anomaly detection etc. Apart from clustering, another key application area where unsupervised learning is used most often is to clean and simplify the data without losing its original characteristics. This concept is known as dimensionality reduction. It can be described as a process of scaling data having a large number of dimensions (features) into fewer dimensions without any change in the data characteristics. It also provides an efficient method to detect anomalies or irregularities in the input data-set.

Semi-supervised Learning: Only a portion (usually small) of the training data is labeled. Obtaining labeled data for training is often expensive and time consuming. Semi-supervised learning helps to create a trade-off between supervised and unsupervised learning. Let us take an example of an application that categorizes web pages depending on their content. Supervised learning would mean manually annotating millions of web pages. A better alternative is to cluster similar pages based on an unsupervised algorithm, and then label each of those clusters



(a) Classification **(b)** Regression

Figure 2.2.: The difference between classification and regression. While classifications predict the associations to a particular class, regression techniques estimate a continuous value.

depending on their types.

Reinforcement Learning: The most ambitious type of learning, where the system or an agent studies its surrounding environment to select the next set of actions. These actions are either rewarded or penalized, and based on those the system learns the best approach to maximize the rewards.

We can further distinguish between the ML methods on the basis of how it can learn from the input data. In many cases the system might have to be trained beforehand with all the available data and then implemented on to the production environment. This type is called **batch learning** or **offline learning**. One limitation of this being the inability to adapt to new data in real time (with every new data the whole model needs to be trained from scratch). Also it can be computationally expensive depending on the size of the data-set. The alternative to this is incremental training of the model with small chunks of incoming data, so that it can quickly adapt to the changes in data characteristics, a concept known as **online learning**. This, while solves the previous bottleneck of high resource utilization, can severely degrade the performance in case the training data collected is erroneous due to even a momentary failure of the data acquisition tools.

Training on the input data can be trivial, where the model will simply memorize learning samples and use them for prediction. This type of learning, known as **instance-based**, will only work if the test data is very similar to the one used for training. A better alternative will be to create a model of the train set, and then use that to make predictions on the new test data. This is known as **model-based learning** [18], and seems to generalize better with respect to data variations.

Machine Learning algorithms can be seen as an enabler, but it is by no means a panacea. In fact, only a fraction of real world projects consist of it. Just like traditional programs, ML algorithms also depend on the quality and quantity of the training data. Typically, for a ML algorithm to have a certain acceptable accuracy we need a lot of data for training, even for the most simple tasks. The model should also generalize well to a broad range of test cases, which requires the training data to be representative. The quality of the training data is also as, if not more important for the ML algorithm to make sense. Careful measures should be taken before, during and after the data collection to make it free of noise or missing values. Perhaps the most crucial aspect of any ML is the feature selection. A clear, well-structured and distinguishable set of features will largely help improve the performance. This whole process of selecting important features, combining or extracting new features out of existing ones is called feature engineering.

Having only a good data-set does not always guarantee performance, a good algorithm is just as important. We often encounter problems where the algorithm performs well on the training set, however the same performance does not reflect when using real life test data for prediction. This problem, often a side effect of the model being too complex, is known as **overfitting**. On the other hand, selecting a model that is too trivial can result in it not being able to understand the intricacies of real world data, termed as **underfitting**. A good ML algorithms always thrives to have a balance between these two, and often depends on the characteristics of the data it is acted upon.

2.2 Class Imbalances

One of most fundamental and often inevitable problem associated with real world machine learning tasks is the class imbalance. The imbalanced learning problem is observed when there is an unequal distribution of classes in a data set. The order of the imbalance however can vary depending on the applications. For example in a medical test conducted on 1000 patients to detect a certain rare disease, only 20 may test positive while the large majority will have negative results. We encounter this problem in real world data more often than not. Imbalanced data can significantly weaken the performance of most standard learning algorithms, and as a result has become a fundamental research area in the field of data science and machine learning. Several solutions has been proposed over the years, we will look into some of them and choose one suitable for our data set.

Common evaluation metrics such as the accuracy score can give misleading outcomes when evaluating training algorithms for imbalanced class data-sets. In the above example of the medical test, just by random guessing we can assume the probability of the patients not having the disease to be around 98%. This would make a classifier score of say 0.99 to be misleading, for in reality it is only 1% better than a random guess. A more accurate metric can be the Receiver Operating Characteristic (ROC) curve [41], which is generated by plotting the true positive rate against the false positive rate for all possible classification thresholds. It is commonly used to visualize the performance of a binary classifier. We take into consideration the area under the ROC curve, known as AUC to measure the performance of our classifiers. A very poor classifier (no better than random guessing) will have an AUC of 0.5, while for a perfect classifier it will be close to 1. The advantage of AUC over normal accuracy metrics is that it is immune to class

imbalance and only cares about how well the classifier separates the two classes. Some other metrics that can be used are Precision-Recall (PR) curves and cost curves. We will look into these metrics in much more detail in the next chapter.

Garcia et al.[23] classified the imbalances into various types. Imbalances are most often due to the inherent characteristics of their data-set. For example, any medical data-set containing test results for a disease will be skewed towards negative outcomes. Imbalances of this type are known as intrinsic. However, they can also be extrinsic in nature. It can very well be possible that the original data space is balanced, but the time and frequency of data collection has rendered the obtained data set to be imbalanced. Besides, we can also distinguish between relative imbalance and absolute rarity. Relative imbalances are usually seen in large data-sets. For example, in a data-set containing 10 million samples, the minority class may contain 10,000 points. We can see that the minority class is neither small nor rare in itself, but only so in relation with the majority class. On the contrary, some data-sets might have very rare occurrence of minority class examples, irrespective of the data collection duration.

Many solutions have been proposed over the years to address the problem. Garcia et al. and Gustavo et al.[6] in their respective articles suggested some methods for imbalanced learning, broadly divided as: 1. Sampling methods, 2. Cost sensitive methods, 3. Kernel-based methods, and 4. Active learning methods. We will only cover the sampling methods, to keep it brief and relevant to our use case.

Random oversampling/undersampling
This is the most basic of the sampling methods. In oversampling, we add a set of replicated data sampled randomly from the minority class. Similarly for undersampling, we remove data randomly from the majority class. In both cases the result obtained is a somewhat equal distribution of both the classes. Albeit simple, random sampling method is not free from error or bias. Removing chunks of data from the majority class may severely downgrade the classifier performance. On the other hand, simply adding replicated data to the minority class may cause the data set to be too specific and uniform in nature, thus leading to overfitting.

Informed Undersampling
The loss of information in simple undersampling and its consequences have been addressed by Liu et al.[31]. Their methods make use of the ensemble learning systems by creating multiple classifiers with different subsets of the majority class. The accuracy of the classifiers is used to determine which subset of the majority class is best suited for use. Alternatively, K-nearest neighbour (KNN) classifiers are also utilized. The classifier is used to select only those majority class examples that are at a certain distance from a certain number of minority class examples.

Synthetic sampling with data generation
Chawla et al.[9] proposed a technique known as the Synthetic Minority Oversampling Technique (SMOTE). They argued that instead of only undersampling the majority class, a combination of both undersampling and oversampling the majority and minority classes respectively can result in better classifier performance. The oversampling is achieved by creating artificial minority class examples. Instead of randomly replicating data for oversampling, it uses K-nearest neighbours to synthesize data for the minority class. One drawback to this method is the overlapping

between classes caused by using same number of artificial samples for each minority class example. This can be addressed using Adaptive Synthetic Sampling [22] methods that can assign weights to different minority class examples.

Sampling with data cleaning techniques
Several methods have been employed to mitigate the overlapping between classes that we encounter in sampling methods. One popular data cleaning technique is the Tomek links [52]. These are basically a pair of data points belonging to opposite class pairs, spaced at a minimum distance from one another. As a result the Tomek links can be used to form the border between the two classes. In the next step all the links can be removed until the minimal distance pairs belong to the same class, which minimizes the overlapping.

Other sampling methods
While the previous methods of sampling are more generalized, cluster-based sampling algorithms can also be used. One example is the cluster-based oversampling (CBO) [25] method, which utilizes K-means clustering techniques. It gives the flexibility to be tailored to more specific use cases.

Another method is the combination of sampling methods with ensemble learning strategies. A common approach is to integrate synthetic sampling with boosting algorithms. The idea is to use a different set of synthetic sampling at each boosting iteration. This ensures the final voted classifier will have a distributed representation of the minority class.

2.3 Evaluation Metrics

In this chapter we will introduce some terminologies used commonly for the purpose of evaluation in machine learning. The primary objective of any ML algorithm is to maximize the probability of correct predictions. For a binary classifier, this means two clearly defined opposing propositions. A prediction for a new instance is correct if it matches with the actual value for that instance, otherwise it is deemed incorrect. This can be represented by a $0-1$ loss function, where 0 means correct prediction and a loss of 1 means an incorrect prediction. A more refined model might include a probability to each prediction, which provides opportunity for further insight into the assessment of the prediction. The quadratic loss function (QLF) is a useful metric often used to evaluate such predictions. It is expressed as:

$$\sum_i (p_i - a_i) \tag{2.1}$$

where $p_1, p_2 ... p_k$ are the probability vectors of the different outcomes for a new training instance, and $a_1, a_2 ... a_k$ is a vector with the only one, suppose the n^{th} component being the actual outcome (assigned 1, the rest being 0). From the equation we can clearly observe that the quadratic loss function takes into account both the probabilities of the true event (that happened in reality) and all the other events, and will depend on the distribution of the probabilities assigned to each event. An alternative to this is the information loss function, expressed as $-log_2 p_i$, which represents the actual outcome i in terms of the probability distributions

$p_1, p_2 ... p_k$. Unlike the QLF, it only considers the probability assigned to the actual outcome. This however, leads to a high cost if the probability assigned to the actual outcome is very low, and in case of 0, will render the function useless.

Confusion Matrix (CM)
One of the most common layouts for performance visualization in binary classifiers is the confusion matrix. It is simply a square matrix that lists the predicted outcomes against the actual outcomes, as shown in Figure 2.3. Below we introduce some terms that can be derived from the CM, and in general describes the performance of a classifier.

1. True Positive (TP): When both the predicted and actual outcomes are positive.
2. True Negative (TN): When both the predicted and actual outcomes are negative.
3. False Positive (FP): The outcome is predicted as positive, while the actual outcome is negative.
4. False Negative (FN): The outcome is wrongly predicted as negative, the true outcome being positive.

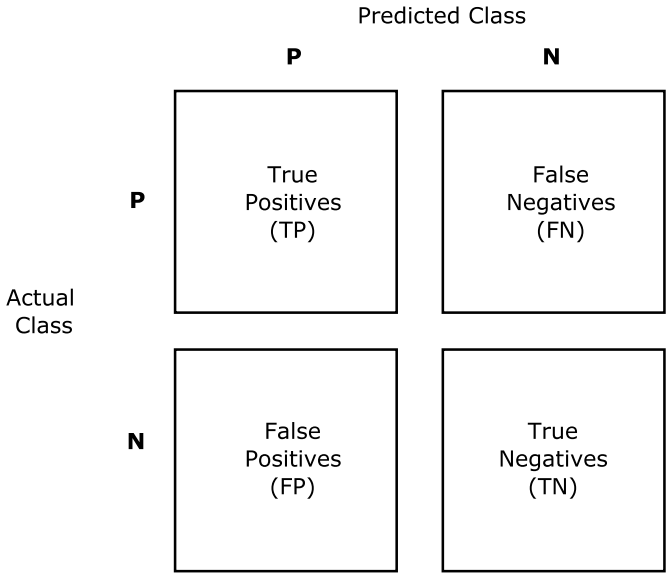


Figure 2.3.: Confusion Matrix for a binary classification.

True positives and true negatives indicate the instances that are classified correctly, and an accurate model is expected to have a higher distribution of both. While it is desired to have as few instances as possible to be false positives (also known as type-1 error) and false negatives (also known as type-2 error), in reality this largely depends on the use case and the objective of the application. An acceptable level of balance for FP and FN must be decided depending on the parameters the model is based on. Let us take an example in cancer diagnosis. A false positive (incorrectly classified as malignant) has a much lesser consequence than a patient incorrectly detected as not having cancer. This can be represented by a slightly different kind of confusion matrix, known as cost-sensitive classification. Different costs can be included for the different types of errors (FP and FN), or different types of correct classifications (TP and TN) can have different rewards. For the cancer example, a higher cost can be assigned with the false negative

samples, which will affect the accuracy of the classifier based on the cost matrix. A highly advanced learning model may also include the cost associated with data collection, training etc. Coming back to the generalized CM, the overall accuracy (also known as success rate) and error are used to gather information about the misclassified samples. The former is calculated as the ratio of the sum of all correct predictions to the total predictions, while the latter is the ratio of the sum of all false predictions to the total predictions. From the equations below we can see that $Accuracy = 1 - Error$ and vice versa.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

$$Error = \frac{FP + FN}{TP + TN + FP + FN} \tag{2.3}$$

True positive rate (TPR) and true negative rate (TNR)
We can derive some other specific metrics from the above equation. A true positive rate (TPR) is utilized to get the number of positive instances that are correctly classified out of the total actual positive examples. Similarly the false positive rate (FPR) is used to obtain information about the ratio of samples classified wrongly as positive to the total number of negative samples. Both are expressed as:

$$TPR = \frac{TP}{FN + TP} \tag{2.4}$$

$$FPR = \frac{FP}{FP + TN} \tag{2.5}$$

True positive rate (TPR) is also called sensitivity, while the inverse of the false positive rate (FPR), or the true negative rate (TNR) is known as specificity.

Precision and Recall are the measures of relevance. Precision can be defined as the ratio of instances correctly identified as belonging to the positive class (TP) to the total instances predicted as belonging to the positive class (TP + FP). Another measure called recall (also known as sensitivity) is used in conjugation with precision (and are often combined). The recall can be expressed exactly in the same manner as the TPR, and in principle are identical to each other.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

The F1 score incorporates the precision and recall into a single metric. It is essentially the harmonic mean of the two. An important thing to note that since it calculates the average based on the reciprocals (unlike regular mean), lower values will have a higher representation. This ensures a fair assessment of the model performance, since it can only achieve a high F1 score if both precision and recall are high [18]. It is expressed as:

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{2.8}$$

An ideal score of 1.0 precision for an outcome can be inferred as every instance predicted to belong to a certain class does indeed belong to that class, however it gives no information about the number of outcomes incorrectly predicted to belong to the other class. On the other hand, a perfect 1.0 recall would mean all the instances from a particular class are predicted correctly, but does not indicate how many instances are incorrectly predicted as belonging to that class. Just like the TPR and FPR mentioned above, these are complementary to each other, and a trade-off is often achieved depending on the use case. A precision vs recall curve is a useful method for selecting the threshold value or the desired balance.

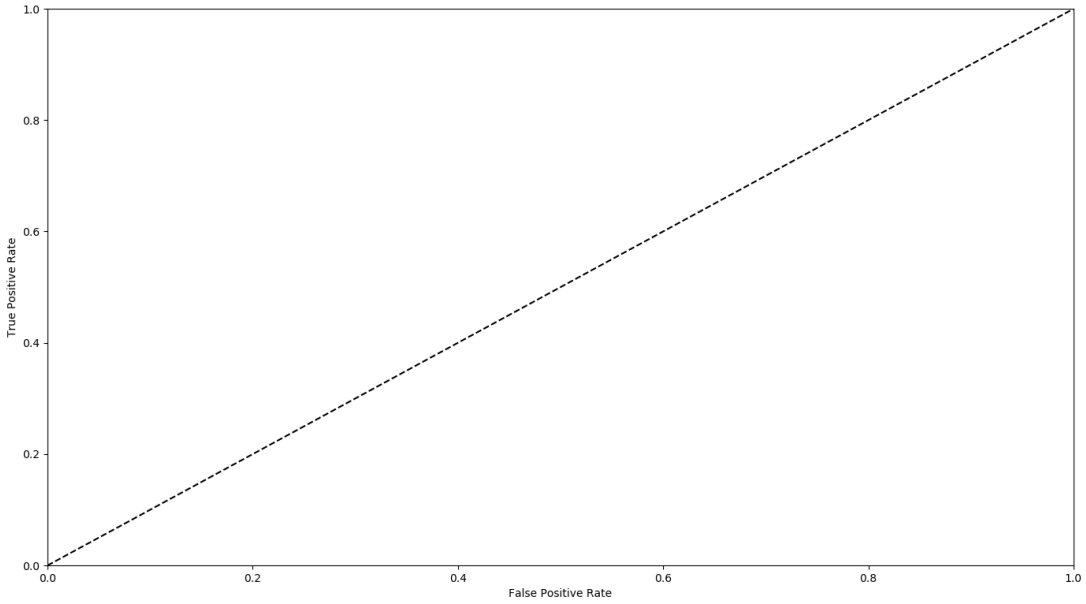


Figure 2.4.: The base structure for a ROC curve.

Receiver Operating Characteristic

The receiver operating characteristic (ROC) [41] slightly differs from the precision-recall curve. Instead of precision, it plots the FPR (1 - specificity) against the TPR (recall). The above Figure 2.4 shows a base structure of a ROC curve. The dotted diagonal can be seen as random guessing. Training algorithms falling below the line is considered worse than random guessing, while an ideal model with TP and FP being close to 1 and 0 respectively should occupy the top-left corner. Each point on the curve can be interpreted as sensitivity/specificity pair for a particular decision threshold. Another metric, known as the area under the curve (AUC) is often used in conjugation with ROC. As the name suggests, it measures the area that falls under the ROC to give a quick visual interpretation of the classifier performance. The properties of a ROC curve can be summarized as follows:

1. Shows the trade-off between TPR (sensitivity) and FPR (1 - specificity) for a particular outcome.
2. The accuracy of an outcome is judged by the proximity of the curve to the upper-left boundary of the ROC space.
3. The diagonal across the centre depicts an accuracy of 50%.

4. The area under the ROC curve is a commonly used metric for accuracy.

In Chapter 4.3 we will take up how ROC curves are an effective performance measure for imbalanced classes, where normal accuracy metrics give misleading results. This is because the ROC can reproduce the model performance without any knowledge or consideration of underlying class distribution.

For the cost-sensitive classification where we can assign specific costs or rewards with the elements of the confusion matrix, the ROC curve fails to depict the cost associations. For this reason cost curves are used to portray a model as a straight line that varies with the changes in the class distribution. It is worth mentioning that the metrics discussed above only depicts the performance evaluation for classification tasks. Linear regression models have their own set of evaluations, however they do not come under the context of our research. We have even glossed over or completely omitted some of the classification measures that are quite similar to the ones described above (lift charts, cost curves etc.) for the sake of conciseness.

2.4 Cross Validation and Parameter Tuning

In the previous segment we discussed about the various performance metrics for binary classification problems. In this section we will briefly discuss about the cross validation (CV) techniques associated with ML. While these also fall under performance evaluation, they can be used before/during the model training unlike CM, ROC etc that are used after predictions. As a result these can also be considered as performance enhancement techniques. Instead of training the model and then check the accuracy of the predictions, we can proactively implement performance improvement measures at the time of training. It is generally a common practice is to use cross validation to divide the data into individual subsets or 'folds', and then use the accuracy metrics like ROC to measure the performance of each subset. The performance of any ML algorithm depends on how well it behaves on new instances. Training and prediction on the same data-set will almost always lead to overfitting of the classification model. For this reason the available data is split into distinct training and testing sets which provides a fair assessment of the model with respect to real world usage with unknown data instances.

A very prevalent method in ML, known as the holdout method, divides the data into training and testing sets and keep them separate at all times. However, another important aspect of a model is to identify or tune its different parameters to make it optimum for a given data-set. An even better approach can be to have a separate data-set for the cross validation. Instead of only the training and the test set, the data is further divided into an extra validation set. This extra data-set can be used for tuning the model parameters at the time of training, and also to evaluate the model performance. The selection of the optimal parameters (called hyperparameters) is an iterative process, and once obtained, are used for the final model selection.

Further improvements can be made to the holdout method to make it more robust and less dependent on how the data-set is split. In k-fold cross validation, the holdout method is duplicated k times (without replacement) for k subsets of the training data. Out of the k splits, only one is used for testing while the remaining $k-1$ folds are used for training. A collective sum

of the accuracies of the different folds will be used for the hyper-parameter selection. The split is conducted without replacement to ensure data from different folds does not overlap, which results in a lower variance than a single random split in the conventional holdout method. The value for k is commonly selected as 10, and most of the times provides sufficient optimization enhancements. A large number of folds will result in better optimization and lower variance, but comes at a cost of CPU resources. A further improvement of this method, known as Stratified k-fold CV [27] can be applied for data with class imbalances. Unlike the normal k-fold method which creates folds of equal sizes, it makes sure the splits are proportionate to their representative classes for better portrayal of the original data-set. There are other similar variations to the ones mentioned above like leave-one-out CV that are more exhaustive in nature. However, these are less popular due to the high computation cost associated with them.

In relation to the above, several methods are applied specifically to find a combination of hyper-parameters that will give optimal performance. A simple but exhaustive method is to specify a series of values for each independent parameters. The model will select the best combination of the parameters via a brute force search. This method is known as **Grid-Search CV** [38], and while it gives the best possible combination of hyper-parameters, it often comes at a price of high CPU time and resource utilization. For a wide range of combinations it often becomes impracticable, as a result a less effective yet resource friendly method, called **Randomized-Search CV** is used. It is similar in its approach to the Grid-Search, however instead of iterating over all possible combinations, it randomly selects a parameter value for each iteration. Surprisingly, it is often found to give very similar optimization results as Grid-Search CV, which makes it a slightly preferred choice for a majority of cases, specially the simple ones. It is also worth mentioning that some classifiers have performance optimization built into their implementation, while in some cases several classifiers can be combined to obtain an improved learning model. We will delve into this in the next sections where we discuss the different classifiers used in our experiment.

2.5 Support Vector Machines

Support Vector Machines (SVMs) can be considered as one of the most popular and all-around models for supervised machine learning, suitable for a wide range of learning tasks like linear and non-linear classification, regression and anomaly detection. For binary classification it can be stated as the idea of finding a boundary that best divides the data into the two separate classes. Linear classification address the most fundamental problems where the two classes can be separated linearly, i.e. the classification threshold (known as decision boundary) is either a straight line or a hyperplane. However, often non-linear methods are needed to be employed which involves applying some sort of transformation function to the data to make it suitable for linear separation. A good ML classification algorithm should have a decision boundary that is as far as possible from the nearest instances of both classes on either side. This is easily achieved in SVMs, a concept known as large margin classification. The closest training instances determine the placement of the decision boundary, and are known as support vectors. However, having a wide decision boundary does not augur well for data with outliers, and often a trade-off in the form of parameter tuning is required to find the right balance between a higher margin and correct classification.

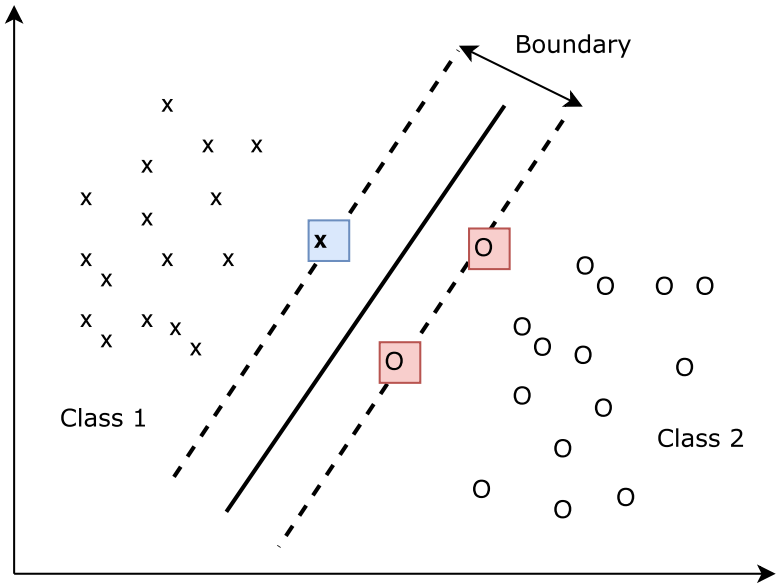


Figure 2.5.: A diagrammatic representation of the decision boundary in SVMs. The coloured data points on either side of the boundary are known as support vectors.

As mentioned above, it may not always be possible to linearly separate the two classes. A common and simple approach is to add more features which may transform the data-set into linearly separable. Although we are applying linear separation on the transformed data, the decision boundary on the original data will seem like a curve that separates the two classes rather nicely. A form of parameter tuning called the kernel method or kernel trick is widely used for non-linear SVM classification, and can be implemented with almost all ML software packages. A kernel can be seen as a function that computes the dot product $\phi(a)^T.\phi(b)$ for the vectors a and b, without having any information about the transformation function ϕ. Some of the commonly used kernels are Linear, Polynomial, Sigmoid and Gaussian RBF. We will discuss briefly about each of them below.

A linear kernel is the simplest approach that creates a linear separation between two classes. This is often ideal for a large but relatively clean data-set without many outliers. On the other hand a polynomial kernel simply adds polynomial features to make the data linearly separable, the degree of which can be optimized according to the complexity of the data. A higher degree will tend to overfit the model, while a lower degree will result in underfitting.

Another way to add new features is to use a similarity function for each instance based on some distinct landmarks. This similarity function is taken as the Gaussian Radial Basis Function (RBF) [8], ranging from 0 to 1. The new features computed from this function will be linearly separable. A tuning parameter, called gamma, is used to alter the shape of the curve for RBF. A small gamma will give a smoother decision boundary, and can result in underfitting. On the other hand, a high value will create a jagged decision boundary very localized to that particular data-set, a clear indication of overfitting.

We conclude by briefly looking into the pros and cons of SVMs. As mentioned earlier, they

are highly effective algorithms when dealing with binary classification of small to medium sized data-sets with little outliers or noise. This can also be stated as a limitation as they are often ineffective for large and noisy data-sets. Another thing worth mentioning is that SVM assumes the input data to be numeric, as a result some data pre-processing is required to convert categorical inputs to some variables. Another important aspect associated with SVMs are feature scaling. A variable with a much higher range of values than others will have a more pronounced effect on the decision boundary. This requires a form of data normalization to convert all variables into a uniform scale, generally between 0 and 1.

2.6 Decision Trees

Decision Tree algorithms are the fundamental building blocks of the Random Forest classifier, which plays a vital role in the model training and prediction of our application data-set. Hence we will look into the fundamental idea of decision trees before moving into ensemble learning, and will explain the various concepts associated with them. Decision Trees can be argued to be an extremely efficient and all-rounded algorithm for training, and scales well with complex and large data-sets. They are represented by binary trees, where each root node corresponds to an input variable x and a point of split for that variable depending on some conditions. The leaf nodes contain an output variable based on which an outcome can be predicted, while the edges correlate to the result of a test and connect to the next node or leaf. The following are the general steps that are taken for the classification.

1. Start at the root node (depth 0).
2. Check the condition.
3. Follow the edge based on the outcome.
4. Repeat steps 2 and 3 until a leaf (terminal) node is encountered.
5. Give the prediction based on the outcome of that leaf node.

What makes Decision Trees (often called classification and regression trees, or CART) unique is that classification of a new example is done by it undergoing a series of tests arranged in a hierarchical structure, and thus the name decision tree. Let us take a dummy example where the input data consists of two parameters height and weight and outputs the gender as male or female. Figure 2.6 shows the decision tree to predict the correct gender for a new example. Each node has three main attributes namely samples, value and Gini. The sample attribute gives the number of training instances. In the above example, 100 training instances may have heights lesser than 75 inches, out of which 54 have weights less than 180 lbs. The value attribute indicates the training instances for that node belonging to each distinct class, either male or female. The Gini attribute measures the node's impurity. It comes from the Gini coefficient, which states if we select two items randomly from a dataset that is considered to be pure, then those two items will be of the same class. A node is said to be pure (Gini=0) if all the training instances applicable to it are from the same class. For example, in Figure 2.6, the depth-1 left

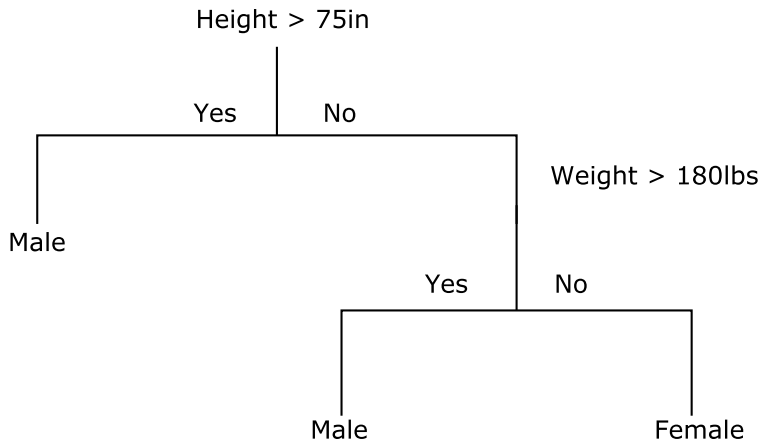


Figure 2.6.: A decision tree structure. Only the nodes that fulfill the conditions are traversed until it encounters a terminal node.

node applies to only training instances labeled male. The equation for Gini impurity is expressed as:

$$G_i = 1 - \sum_{k=1}^{N} p_{i,k}^2 \tag{2.9}$$

Where $p_{i,k}$ is the proportion of class k instances to the training instances in the i^{th} node.

Let us assume Figure 2.7, that shows the decision boundaries for the above decision tree with the number of splits being three. Overfitting is often a key issue when setting the parameters for a tree model. If no limitation is set, it can in the worst case scenario go on creating a separate leaf for each distinct observation, resulting in 100% accuracy for the model. As a result, it is crucial to set the maximum depth of the tree with the help of parameter tuning.

An intuitive observation of the above example indicates the importance of the decision as to when and where make the splits and how it will impact the model's accuracy. A common practice is to split the nodes on all available input parameters and then go on to choose the split that results in the maximum homogeneity of resultant sub-nodes. Different algorithms can be used for the splitting, one of them being based on the Gini index, but is only limited to binary classifications. Some other methods are the Chi-Square (measure of the difference between the sub-nodes with their parent node) and Information Theory (measure of the nodes' entropy).

Decision Tree classifiers are often considered to be a robust algorithm well suited for a large variety of classification tasks, and have several advantages over linear classification models. To begin with they require very little or no data preparation for training, unlike some other models like SVMs. Data pre-processing (standardization, normalization etc.) can be computationally expensive and time consuming depending on the size of the input data. They work well for both numerical and categorical data and is suited to operate on large and complex data-sets.

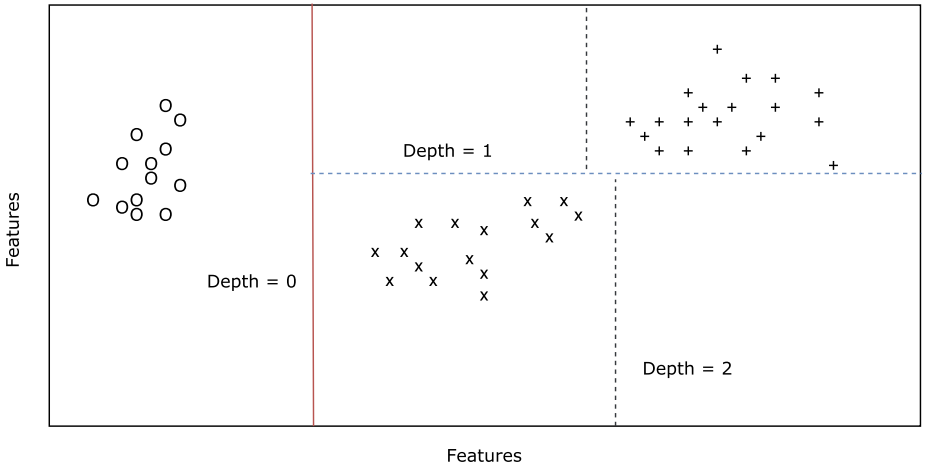


Figure 2.7.: A decision tree with a depth of n=2.

2.7 Ensemble Methods and Random Forests

It is generally argued that a collective opinion regarding a certain solution is usually found to be at par, and often better than a single expert opinion. This concept is the fundamental basis of ensemble learning, which is based upon the principle that if we take the combined results of a group (ensemble) of several weak predictors, it is more likely to perform more accurately than the best single predictor. A majority voting classifier is a simple example of ensemble learning. We train our model using multiple classifiers, and then on a new instance take the aggregate of all the predictions and classify it based on the majority of the votes. A collection of some weak learners (low prediction accuracy) can be combined to create a strong learner with high performance measures.

Unlike the majority voting method where we use different training methods on the entire training data-set, we can use the same decision tree algorithm on the different subsets (seed) of the training data. On paper, it might be expected that it will result in the same tree structure for all the subsets. However, the decision tree algorithm is susceptible to churn for slight variations in the input data, and in reality will have varied predictions for the different seeds. If the random sampling is carried out with replacement (multiple sampling for the same predictor), the method is known as bagging (bootstrap aggregating) as shown in Figure 2.8. Each tree votes on the new instance, and the class with the higher voting is taken as the correct one.

Usually complex learning models tend to have a lower bias and higher variance. A bias or error rate measures how accurately the model matches the problem. Bias is a perpetual property of the learning algorithm, and can never be eliminated completely, even with an infinite number of input data subsets. On the other hand, the the concept of variance is derived from the fact that a particular training set is often not the exact representation of real world data. The total error for a model can be summed up as the combination of both, known as the bias-variance decomposition [55]. The idea of bagging is to minimize the variance by working on different subsets of the same size from the training data, at the same time not compromising the bias too much. An ideal solution to this would be to use independent training sets for each tree, something that is often unfeasible or expensive at the least.

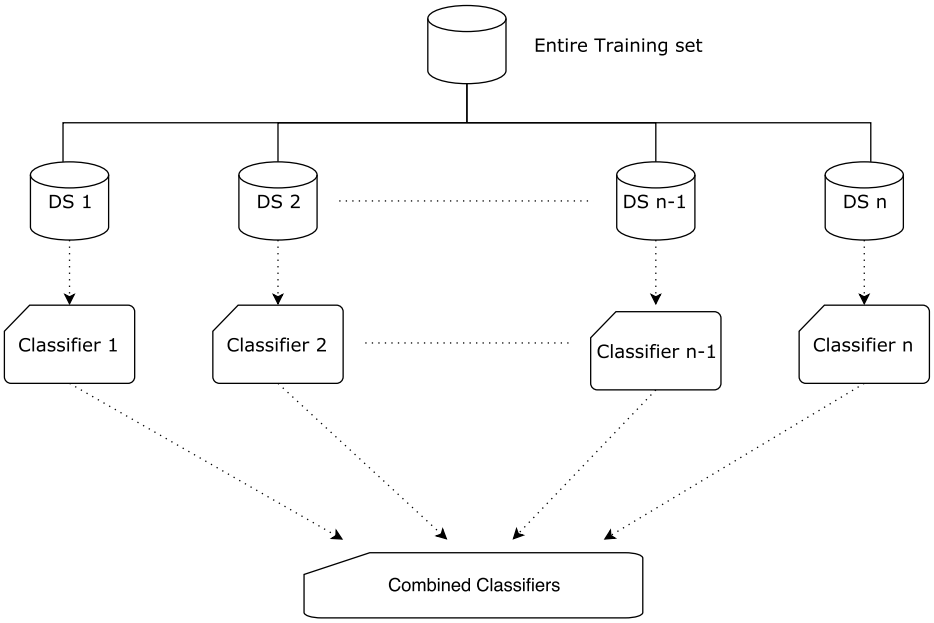


Figure 2.8.: Bagging method. The entire data-set is divided into n random sub-samples with replacement, each of which is trained by a separate model.

We can employ ensemble methods on the Decision Tree based modeling discussed in the previous segment, which tend to have high variance. A Random Forest is an ensemble of several decision trees produced in every iteration of the bagging algorithm. The trees are chosen to minimize the correlation between individual classifiers. A small (but random) subset of features is taken into account before each split, instead of applying the greedy algorithm (minimal cost function) to get the best split point. The algorithm [29] for the random forest is given below:

1. We take N bootstrap samples from the training data. For each of those samples, we construct a usual decision tree. But unlike the normal tree where the best split is decided for all the variables in the training data, we take a randomized sample from the training data and obtain the best split only for that subset. A common strategy to decide the value of N is to keep creating trees until the error cannot be minimized any further.

2. The majority vote among the N samples are considered for the final prediction on new instances.

2.8 Gradient Boosting

Boosting is a type of ensemble learning method which incorporates a group of weak learning algorithms into a strong learner. Weak learning algorithms can be defined as those whose performances are slightly better than random guessing. The first step is to apply weak learning algorithms with a different distribution, which results in a prediction rule. This is repeated for several weak learning algorithms thus obtaining many weak prediction rules, which are finally merged into a single strong rule [13]. The different distributions for each round are chosen based on those examples that are classified incorrectly and are assigned higher weights in the succeeding rounds. The two most popular boosting models are Adaptive Boosting and Gradient

Boosting, and we will discuss briefly about them in this section.

As mentioned above, an AdaBoost [46][13], short for Adaptive Boosting classifier first employs a base classifier (logistic regression or SVMs) to build a model on the training data. For the next iteration, a different distribution is used where the wrongly classified instances are assigned a higher weight. With each iteration the number of misclassified instances decreases, and the final resultant model with a very high accuracy is obtained. The weak classifiers used in AdaBoost are usually decision trees with a single split.

Let us explain the above principle in terms of a binary classification example for a better understanding. For the first step, all the the samples are assigned equal weights and it tries to classify the samples as best as it can. For the next step, it will determine the wrongly classified samples and assign higher weights to each of them, at the same time reducing the weights of the correctly classified ones. The weight is directly proportional to the accuracy of the predictor. So in essence the predictor weights from the previous iteration is used to update the instance weights for the next iteration, with the wrongly classified instances being 'boosted'. This makes the next classification attempt to concentrate on the higher weighted samples (incorrectly classified) and adjust the decision boundary. With enough iterations we would normally end up with a strong classification algorithm by taking the majority for each weak learners. The same or different learning methods can be used for each step, with an option to set the maximum limit for the number of rounds.

AdaBoost takes some assumptions on the training data. The data should be relatively free of noise and outliers, as it would attempt to rectify even those instances at the expense of computation without adding any value to the model. Overall, the data should be of good quality to get the optimum benefits from AdaBoost.

Another popular boosting model is known as Gradient Boosting [14]. The basis of this model is to identify a loss function depending upon the particular use case, and then optimize the loss function through multiple iterations. The weak learners used in this model are typically decision trees, for which the splitting is decided based on the purity of the nodes (Chapter 2.6). The model iterates over the residual errors made by the preceding learners until a desired loss function is achieved. Final prediction on a new instance is made by aggregating the predictions for all the learners in the tree. The contribution of each tree can be tuned as a hyper-parameter, and can be compensated with increased number of trees for a better generalization of the test set. This regularization technique is known as 'shrinkage'. Limiting the decision tree parameters (number of trees, tree depth, nodes and leaves etc.) is another useful trick to enhance the gradient boosting algorithm. Some other forms of the algorithm are Stochastic Gradient Boosting [15] and Regularized Gradient Boosting. These aside, efforts have been taken to make the algorithm scale to extremely large data-sets without compromising the learning speed. We will look into the latter in a bit more detail, since it constitutes one of the classifiers we have used in model training.

Extreme Gradient Boosting (XGBoost) [10] is a highly optimized variant of the gradient boosting model, and provides higher speed and scalability by tuning the algorithm to utilize the hardware resources. It extends portability to various distributed processing frameworks like Apache Spark

[56]. However, adding parallelism and multithreading to existing gradient boosting models is not the only trick applied by XGBoost. Most of the computational algorithms are written in C++, which along with data preprocessing methods built in by default, lowers the execution time to a large extent. The model also ensures that it minimizes overfitting by applying regularization methods. High flexibility, taking care of missing or empty values, robust inbuilt cross validation support etc. are some of the many advantages [33] XGBoost has to offer over its predecessor.

2.9 Probabilistic Learning and Bayes' Theorem

Statistical methods act as a tool for scientific analysis. In our research, we employ probabilistic methods to improve on our predictive models. The motivation for this can be two fold. First we can get away with a largely down-sampled data if we wish to do so. The quality of data collection is influenced by the sensors and its surrounding. It is always desirable to have the option to switch to less expensive data acquisition systems offering lower sampling rates for data collection. On the other hand, weak connectivity between the sensors and edge devices might have detrimental effects on the collection of data. All these uncertainties affect the classifier performance, which can be mitigated by the use of an 'add-on' or attachment model that is based on probabilistic methods. The second advantage lies in the learning models themselves. Several learning algorithms like ensemble learning or boosting methods provide robust algorithms with built in error reduction and enhanced performance. However, these often come at a price of increased computing resources which becomes a bottleneck for low powered devices. Weak learners or single classifiers in isolation can be used for the prediction, which consumes fewer resources but at the expense of accuracy. This can later be rectified using probabilistic inference models.

Bayesian inference (BI) is a way of making guesses about what our data mean, based on, sometimes very little information. It is a way to capture the common sense knowledge about the situation that helps to make better guesses regarding certain outcomes. The foundations of BI comes from the Bayes's theorem, which again employs the basic building blocks of conditional probability. We will briefly introduce some of the concepts needed to explain the Bayesian Inference.

A conditional probability is a probability based on some historical or background information, and is represented as $P(A|B)$. The notation simply means the probability of A given the condition that B is true.

A conjoint (or just simply joint) probability is the probability that two outcomes are true at the same time, which is represented as $P(A and B)$. This simply means the probability of A and B both being true. The previous notation can be expressed as the following equation:

$$P(A and B) = P(A) \times P(B|A) \tag{2.10}$$

For independent events, where the outcome of A does not affect the outcome of B and vice versa, the equation becomes:

$$P(A and B) = P(A) \times P(B) \quad [since\ P(B|A) = P(B)] \tag{2.11}$$

Conditional probabilities are however not interchangeable, i.e. $P(A|B) \neq P(B|A)$, and its often required to find one given the other. The Bayes' Theorem provides a solution, which takes advantage of the commutative nature of joint probabilities. We arrive at the following equation in a few steps:

From the definition of conjoint probability, it can be intuitively argued that, for any outcomes A and B:

$$P(A\,and\,B) = P(B\,and\,A) \tag{2.12}$$

Using this in the relation for conditional and joint probability (equation 2.10), we get:

$$P(A\,and\,B) = P(A) \times P(B|A) \tag{2.13}$$

$$P(B\,and\,A) = P(B) \times P(A|B) \tag{2.14}$$

Substituting the values in equation 2.12, we achieve the final formula as:

$$P(B) \times P(A|B) = P(A) \times P(B|A) \tag{2.15}$$

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)} \tag{2.16}$$

The above equation is known as Bayes' theorem, a simple yet powerful tool for probabilistic learning. A common interpretation of the theorem is based on its diachronic nature [11]. In simple terms, this means we begin with an initial probability of an outcome, and with each new data update the previous probability through an iterative process. The probability of an outcome before new data is gathered is known as **prior probability** or simply prior. It is often based on background information or historical data. For scenarios where little or no background information is available, the initial prior probability can be taken as 0.5, i.e. equal possibilities for all outcomes. The new probability obtained after each iteration is called the **posterior probability**, which becomes the prior for the next iteration. We can rewrite the equation for Bayes' Theorem with respect to the prior and posterior probabilities. For a hypothesis H, and new data expressed as D, the equation becomes:

$$P(H|D) = \frac{P(H) \times P(D|H)}{P(D)} \tag{2.17}$$

where,

- $P(H)$ is the prior probability.
- $P(H|D)$ is the posterior probability.
- $P(D|H)$ is the probability of the new data given the hypothesis, known as the sampling distribution or the likelihood function.
- $P(D)$ is the probability of the data independent of any hypothesis, known as the normalizing constant.

In machine learning, we can build a model from the above formula to further reinforce the predictions on new data. Any Bayesian Inference model can consist of the following steps. The very first steps include understanding the data itself, and to determine the various outcomes associated with it. The probability distribution of an outcome with only two values is known as Bernoulli distribution. A probabilistic model is established to show the different parameters for the data. The second step will be to define the range of probabilities for the hypothesis, and their associated priors. In the final step, we apply the BI to the incoming data to generate the posterior probabilities.

Bayesian methods also have their share of limitations. Historical data can be grossly incorrect at worst and subjective at best. Background information is often interpreted differently, and to make things worse, there can be more than one source of information. Choosing a prior is perhaps the most important step in Bayesian learning. Starting with a bad prior is akin to starting with a bad assumption, and although BI methods have its own mechanism for 'course correction' based on the likelihood function and input data, this adds undesired complexity to the model. Statistical inference methods are in general computationally expensive and time consuming, and careful effort should be made to get the job done with a minimal amount of input data and iterations.

3 Data Exploration

In this chapter, we will spend some time discussing about acoustic signals and their emission properties that represent our data-set.

3.1 Acoustic Signal Emission

Before beginning to look into our raw data we will spend a little time to identify what kind of data we are dealing with. While every piece of data is unique, they can often be categorized into distinct types for which a certain signal processing method can be implemented. Structural changes in a material results in a spontaneous release of the strain energy, where each energy releasing event is known as an acoustic emission (AE). This can be used to have a sensing and monitoring system (of structural integrity) that is passive, continuous, sensitive, non-evasive, real-time and in situ [51]. The AE signals can carry information about their emitted sources, like component materials, size, location and structural modifications.

AE Signals
We can use a variety of signal processing methods to AE signals. This can range from general purpose methods to methods specially designed for AE signals. However, given the diversity and mutability of the signals emitted, it is advised to select a generic non-parametric method that does not make too many assumptions about the nature of the signal. Having said that, the AE signals can be broadly categorized into two classes, continuous and burst signals. The former is usually associated with gradual deformation of materials under the influence of a sustained force. On the other hand, burst signals are short-term emissions generated due to structural damages in the material [43]. While continuous signals might be encountered in some special structural case that exhibits continuous generation, they are mostly a combination of overlapping burst signals. The AE signals, in general is a series of overlapping (sometimes separate) bursts of descending amplitude occurring at irregular time intervals, and is likely to be accompanied by surrounding noise. While a single burst is crucial to obtain the characteristics of the AE signal in detail, it is in itself not complete if the number and rate of all the bursts for an event are not considered collectively.

Signal Processing
For the purpose of condition monitoring, normally a large number of AE sources are deployed at various locations. This leads to the AE signals being a composite one embedded with background noise. A robust signal processing methodology is needed to characterize the signal and obtain meaningful information for analysis. The approaches can be broadly divided into the following.

- Signal enhancement: Minimize the background noise to enhance the signal quality.
- Signal separation: Separate the individual AE bursts for a detailed characterization.
- Propagation channel identification: Determine the wave propagation characteristics.

- Source location and characterization: Identify the various AE sources and their characteristics, which often helps to obtain a-priori knowledge regarding the nature of the AE signals.

Methodology
AE signal processing algorithms can be broadly classified into four approaches, as discussed by Terchi et al.[51]. These are briefly mentioned below.

1. Non-parametric signal processing: A generalized method that can be applied to any signals irrespective of their nature. The method treats the signal as an input stream of data and does not try to guess the parametric characteristics associated with it. The versatility of this method comes with a cost, since it fails to take advantage of the signal characteristics. Examples of this methods include digital filtering or wavelet transformation.

2. Model-based signal processing: These methods assume a parametric model of the signal, which tries to find correlations or patterns in the signal to predict its changes over time.

3. Statistical signal processing: Sometimes for complex signals a non-deterministic approach is required and the signal is described in terms of its statistical characteristics.

4. Non-linear signal processing: For signals with non-linear characteristics, a different approach is needed to take into account the chaotic behaviour associated with them. Innovative approaches including chaos theory is tried out for the signal analysis.

3.2 Data Collection Methodology

In this segment we will briefly look into the experiment setup for our raw data collection. Although not directly related to our task (since we did not do any data collection ourselves), some background information about the data we are working with is always relevant for understanding its nature. Our main goal is to provide an end-to-end condition monitoring solution on an existing industrial setup, and it makes sense to spend some time for exploration of the setup itself. The primary components of the test-bed are the fluid tanks, each of which is equipped with four hydrophones. From now onward we will use the words tanks and reservoirs interchangeably to convey the same meaning. A leakage in the tank wall would result in air bubbles seeping inside. The purpose of the hydrophones is to detect air bubbles inside the tank. Pre-amplifiers are connected to the hydrophones to enhance only the frequency content relevant to our test case. This ensure the resultant signal to be relatively free of noise. The signal is next collected using a data acquisition system (DAQ) for further processing. Figure 3.1 shows the schematics of a single test-bed.

Hydrophones
Careful consideration was needed for the choice of hardware. For small leaks it is imperative that the air bubbles generated would also be small, thus exhibiting very low acoustic emissions. Hydrophones with relatively high sensitivity are required to capture this information, while being small enough to fit inside the tank. On top of that, the hydrophones should maintain their structural integrity against pressure and corrosive nature of liquids inside the tank. The devices chosen for the setup exhibit high sensitivity and signal to noise ratio (SNR) which gives the

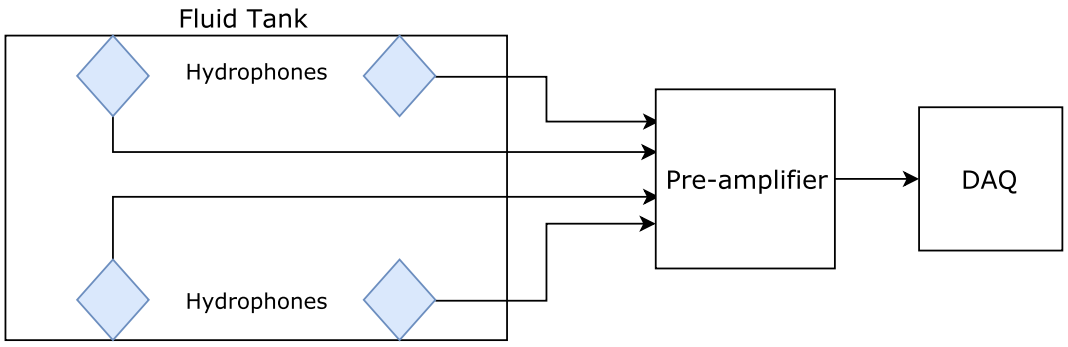


Figure 3.1.: A test-bed for the data collection. Four hydrophones are connected to a pre-amplifier for signal enhancement. The DAQ collects the input from all the hydrophones for further signal processing.

added flexibility of using extension wires if required without any considerable decline in sensitivity. Since each of the four hydrophones is placed at different locations inside the tank, it is beneficial to have the flexibility of calibrating each device individually if and when the need arises.

Pre-amplifiers

To complement the above, the pre-amplifiers should have high noise tolerance. This can be achieved with high gain and the flexibility to choose the high pass cut-off frequencies based on the environment. For our setup two separate hardware were tested for the preamps, with different capabilities and price range. The first one offers the option of operating at two different gains of 10 dB and 30 dB and multiple high pass filter ranges. Although they come at a low price, each preamp can only connect to one hydrophone. This means in our case we need four preamps for each reservoir, thus nullifying the cost effectiveness to some extent. The second one offers enhanced functionality but at a higher cost. An added advantage is the signal being maintained in its analogue nature, thus preserving all the characteristics. The device comes in 1, 2, 3, and 4 channel configurations to provide specific channels to each sensors to prevent interference. This means a single 4-channel setup is sufficient for four hydrophones in a single tank. It also provides seamless connectivity in the form of serial control interfaces. A judicious approach is needed to choose between the two, with performance and cost-effectiveness being the two significant parameters.

Data Acquisition Systems

The DAQ selection depends on both the hardware and the experiment parameters. The sampling rate will rely on both the hydrophone bandwidth as well as the bubble frequencies. The DAQ selected for our setup has a high sampling frequency of 100 KHz with 16-bit resolution, and supports four channels for concurrent data collection. A collection of four DAQs can be combined to make up the DAQ platform, and provides fast downlink connectivity (DAQ to sensors) through modular I/O and uplink (DAQ to CPU) via USB. They are also structurally secure, with wide operating temperature ranges and resilience to vibration and shock.

3.3 Raw Data Analysis

Once the data is collected, we need to find some ways to summarize the raw data. The first steps in any machine learning tasks require a minimum understanding of the acquired data-set. The way data looks in its authentic structure is an essential foundation for the next steps where we extract the important features from the data and apply descriptive statistics for further interpretation. In this segment we will study our data-set as it is, before we apply feature engineering to obtain the important characteristics.

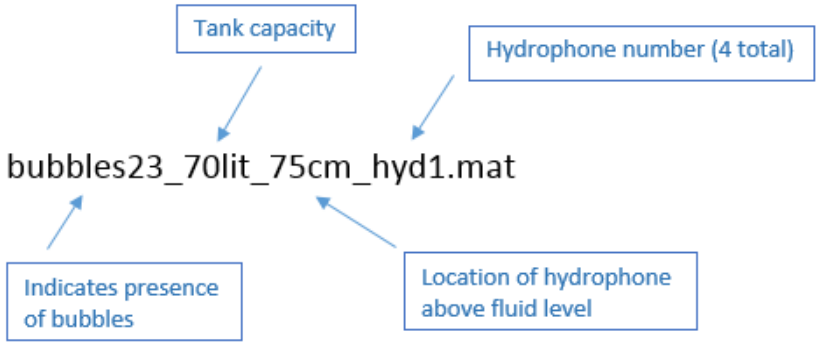


Figure 3.2.: Nomenclature of a random block from the data-set.

The data had been collected on separate independent blocks at different times. From now onward we will refer to each block as a single specimen of our experiment. Each specimen is likely to represent individual reservoirs. We have a collection of 127 such specimens making up our entire data-set. Each block is a binary file containing raw sensor data of exactly 120 seconds duration sampled at 100 KHz. The filename assigned to each block or specimen gives a clear indication of the nature of the data collected. For example, Figure 3.2 shows the various information that can be obtained from a file. We can see whether the data has been collected from a faulty tank and of which capacity, the microphone that recorded the signals and its distance from the surface of the tank. This information will be vital during feature extraction, and also for creating metadata.

It is worth remembering that the entire experiment is conducted in a controlled environment with the outcome of each specimen generated in a predetermined manner. This makes it possible to treat the whole thing as a supervised learning problem. We take two random specimens of data, one each from the operative and faulty tanks for our data analysis. Although each block is independent of one another with different temporal references, we can still compare their displacement characteristics. Figure 3.3 shows the two signals for the first 25,000 samples both in time and frequency domain. In the raw form, we can see that they are indistinguishable from each other. The frequency spectrum indicates the signal is relatively free of noise or outliers, with the peak frequency being at around 50 Hz. In Chapter 4.4 we will use this information for resampling our input data, and further evaluate how it affects classifier performance and resource consumption in Chapter 7.3. We can also infer the data to be symmetrical and evenly distributed, with the median value at 0. This rules out the need to implement noise reduction

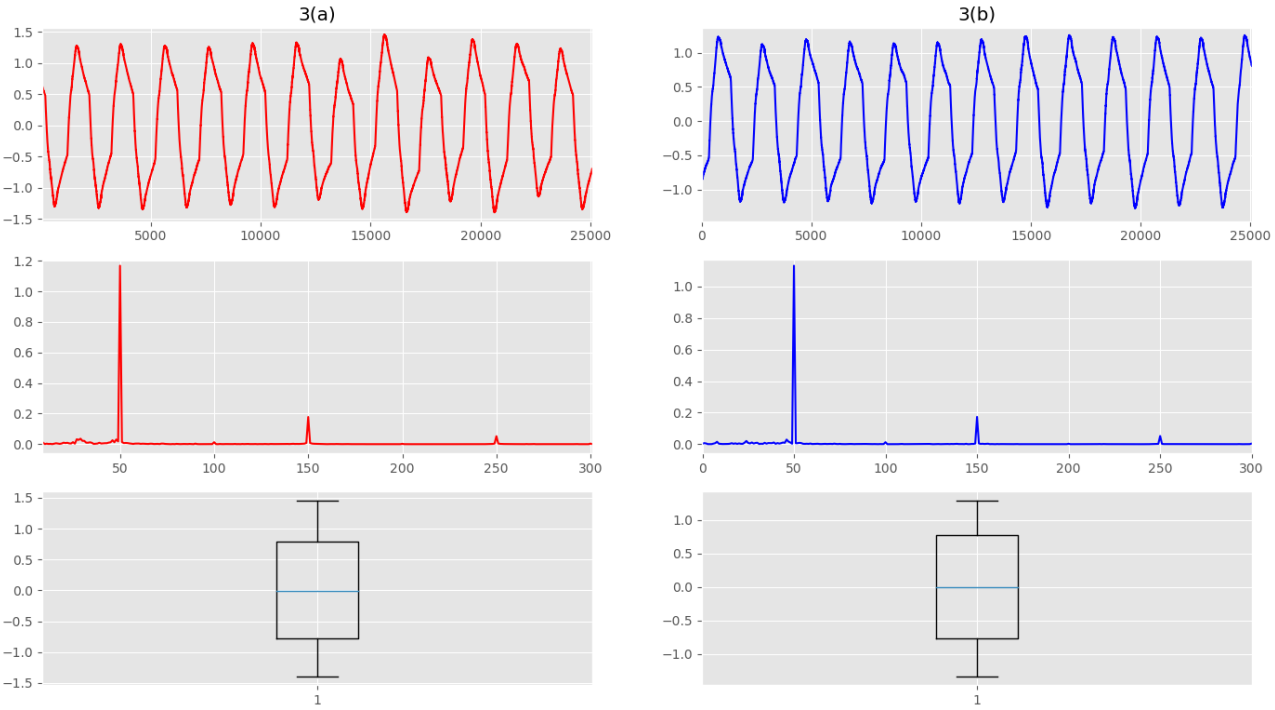


Figure 3.3.: Time and frequency domain characteristics for specimens containing (a)bubbled data and (b)non bubbled data. We can observe that the two classes of data are inseparable from each other in its raw form.

algorithms on the data. It can be argued that it might improve the signal quality even further, but it is hard to justify introducing additional computation for minor improvement. On top of that, certain ML algorithms like ensemble and boosting provide a good deal of robustness to handle data with a few imperfections, and can be further improved with probabilistic models of inference. Next, we proceed to a more extensive analysis of the data-set followed by extracting the features for the training model.

4 Feature Engineering

The current chapter deals with extracting features from the data-set, followed by a discussion about the imbalanced class problem and signal quantization techniques.

4.1 Features Extraction

In the previous chapter we studied the acquired data in its raw form and reached two significant inferences. A top level analysis tells us that the data is relatively free of noise, which saves us the effort to implement some noise cancellation mechanism. The bad news is that the two classes of data, in its raw form, are inseparable from each other. For any algorithm including ML, the quality of the input generally decides the quality of the output. This alone justifies spending considerable amount of time to scrutinize the data. But before we can proceed with any meaningful data exploration we need to go through a process called feature engineering. It can be seen as a way to transform the raw data into features that are more suitable for predictive models to interpret, ensuring improved classifier performance [35]. Good features complement the predictive models they are applied to, and often minimizes the need for high performance models. Feature engineering does not have 'one size fits all' solution, and is often an iterative process which involves repeated model evaluation of the extracted features until no further improvements can be made, or the application has reached its target performance levels.

As discussed in the previous segment, our data-set consists of individually collected specimens each of 120 seconds duration, with the sampling frequency being 100 KHz. This means 1 second of data would contain 100,000 samples, which from now onward we will refer to as a batch. The total number of batches will depend on the time interval we choose for a single batch, also referred to as batch size. For the above example this would mean for each specimen, if we take the batch interval as 1 sec it'll give us a total of 120 batches. We will compute our features over these individual batches. For the features we intuitively decided to use the time domain characteristics of the signal, which points out the variations of amplitude of a signal with respect to time. The reason behind this is our assumption that the signal can best be statistically described in its time domain, a claim which we will later verify using our classifier performance. Below is the list of features extracted from each batch of data.

1. Mean: It is a measure of central tendency of a distribution. For a given set of values $x_1, x_2, ..., x_n$, the mean can be expressed as:

$$\bar{X} = \frac{1}{N} \times \sum_{j=1}^{N} x_j \tag{4.1}$$

2. Standard Deviation: Variability is the measure of the variations of a distribution from its central value. It determines the spread or width of the distribution. For the same set of

values, it is expressed as:

$$Var(x_1, ...x_n) = \frac{1}{N-1} \times \sum_{j=1}^{N} (x_j - \bar{x})^2 \tag{4.2}$$

Standard deviation is the square root of variance, usually denoted by σ:

$$\sigma(x_1, ...x_n) = \sqrt{Var(x_1, ...x_n)} \tag{4.3}$$

3. Skewness: Also called the third moment, it is the degree of asymmetry of the distribution about its mean. It can be considered as a non-dimensional property, since it only describes the shape of the distribution and does not depend on the how the elements of the data-set is measured. It is given by:

$$Skew(x_1, ...x_n) = \frac{1}{N} \times \sum_{j=1}^{N} \left(\frac{x_j - \bar{x}}{\sigma} \right)^3 \tag{4.4}$$

4. Kurtosis: Also known as the fourth moment, it measures shape of the tail of the distribution with respect to a normal distribution (zero Skewness). It can be further classified as mesokurtic, leptokurtic and platykurtic based on the nature of the tails. For the same distributions as above, kurtosis can be expressed as:

$$Kurt(x_1, ...x_n) = \left(\frac{1}{N} \times \sum_{j=1}^{N} \left(\frac{x_j - \bar{x}}{\sigma} \right)^4 \right) - 3 \tag{4.5}$$

5. Root mean square (RMS): Also called the quadratic mean, it is the square root of the mean values x_i^2, and is expressed as:

$$RMS(x) = \sum_{j=1}^{N} \left(\frac{x_j^2}{N} \right) \tag{4.6}$$

6. Peak to peak (P2P): It is the measure of the difference between the maximum positive (peak) and the maximum negative (trough) amplitudes of a wave. It can be expressed in absolute terms of the amplitudes as:

$$P2P = |amplitude_{max}| + |amplitude_{min}| \tag{4.7}$$

7. Crest Factor (CF): It is the measure of the ratio of the peak amplitude to the RMS of a waveform, usually expressed in dB as:

$$CF = \frac{x_{peak}}{x_{RMS}} \tag{4.8}$$

Apart from these time domain characteristics, we decided to add two more features as they can be considered to be important parameters for our data-set. We obtain these attributes from the nomenclature of the individual test samples we came across on Chapter 3.3.

8. Tank capacity: The samples are collected over tanks with three different capacities of 32 liters, 62 liters and 70 liters. An AE signal has a very strong correlation with its surrounding environment and the dimensions of the enclosed space can have varying effects on the signal characteristics. For this reason we have considered the capacity of the tanks as one of the features for the data-set.

9. Microphone distance: Another piece of information we can extract from the naming of the test samples are the id and distance of the hydrophones from the base of the tank which is responsible for recording that particular specimen. The distances are found to be 5 cm and 75 cm, which although small enough not to cause any measurable change in signal characteristics, can still be considered as one of the attributes. This might not have any relevance to our current setup, but can be an important aspect for bigger tank sizes with multiples hydrophones spaced evenly or at irregular intervals.

Now that we have extracted the relevant attributes from our data-set, the next logical step is to identify the target (output) variables, which would let us treat our models as a supervised learning problem. The target variables, or labels can be obtained in the same way as the last two features. Each of the specimen names start with either "bubbles-", or "no_bubbles-" which provides information about the nature of the data collected. We assign the value of 1 for those having bubbles, and 0 otherwise. This a common practice in ML for binary classification problems, where the primary objective of the application decides what should be treated as the preferred class (label 1).

Before moving into the next section, let us briefly examine the data-types of the variables we are dealing with. This information will come handy at the time we employ descriptive statistics for the data, as different forms of data often demand its own analytic methods. The variables can be grouped into three different categories, depending on how it is represented in the data-set, the data type and the nature of the variable. For the first part, the variables can be divided in terms of features and labels, or in simpler terms into input and output variables. An inspection can also tell us the data-type it represents, like whether we are dealing with numeric or text data. Lastly, the continuous variables can be distinguished from the categorical ones, as in the case of microphone distance in our data-set. Continuous variables are best understood by examining their central tendencies, while for categorical values we are normally interested in finding out the distribution of each category.

Not all features are created equal. Some features will have a larger influence on the predictive capacity of the model than others. This can be verified with the help of a Pareto chart [54] as shown in Figure 4.1. We obtained the feature importance using the Random Forest classifier. The bars represent the individual features and their corresponding importance in the prediction process, while the curved line represents the cumulative sum. We can notice an additional feature ‘random’ in the figure. As the name suggests, this feature consists of randomly generated numbers that has no relation to our existing problem or the data-set. The reason behind creating this temporary variable is to have a simple way to check if any of our generated features is

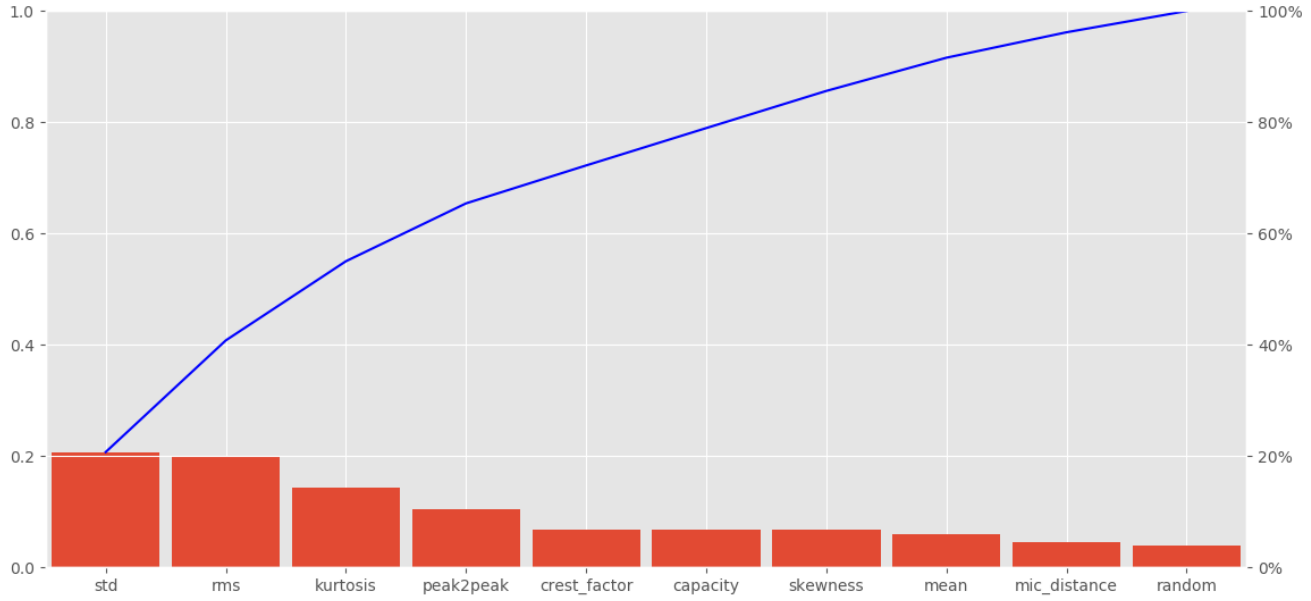


Figure 4.1.: Pareto chart showing the importance of the different features for training with the random forest model.

performing equal to or worse than this random feature. If yes, then that feature can be deemed worthless and we can simply remove that from our list. Indeed we can see that all our features are better than this variable, although some by only a small margin. We can also notice that both standard deviation and root mean square are the most significant features in our data-set for the RF model.

4.2 Descriptive Statistics

In the previous chapter we analyzed the raw data obtained from the DAQs before proceeding with the feature engineering. While it gave us a basic insight into the characteristics of the data in its original form, a deeper understanding is required for the features both individually and in a collective manner. Another way to look at this is that there must be changes in the data characteristics following the feature extraction. The original data has been segmented and transformed into a summary of its time series statistics. It is important to capture and document these changes to better understand the dynamic nature of our data, and will help in evaluation of our models (from the perspective of generalization) at a later stage when we decide on the best possible combination for our use case.

We begin with analyzing each feature separately and independent from each other. This is known as univariate analysis, and is one of the most fundamental methodologies for statistical data analysis. Figure 4.2 shows the shape of the data distribution for each attribute in terms of Kernel Density Estimation (KDE). We first take a look at the categorical variables which are the tank capacities, microphone distances and the labels of each training instance. We can clearly see from the density plots that each of these variables take distinct values, and can be grouped as categorical univariates. For the microphone distance we can notice somewhat even distribu-

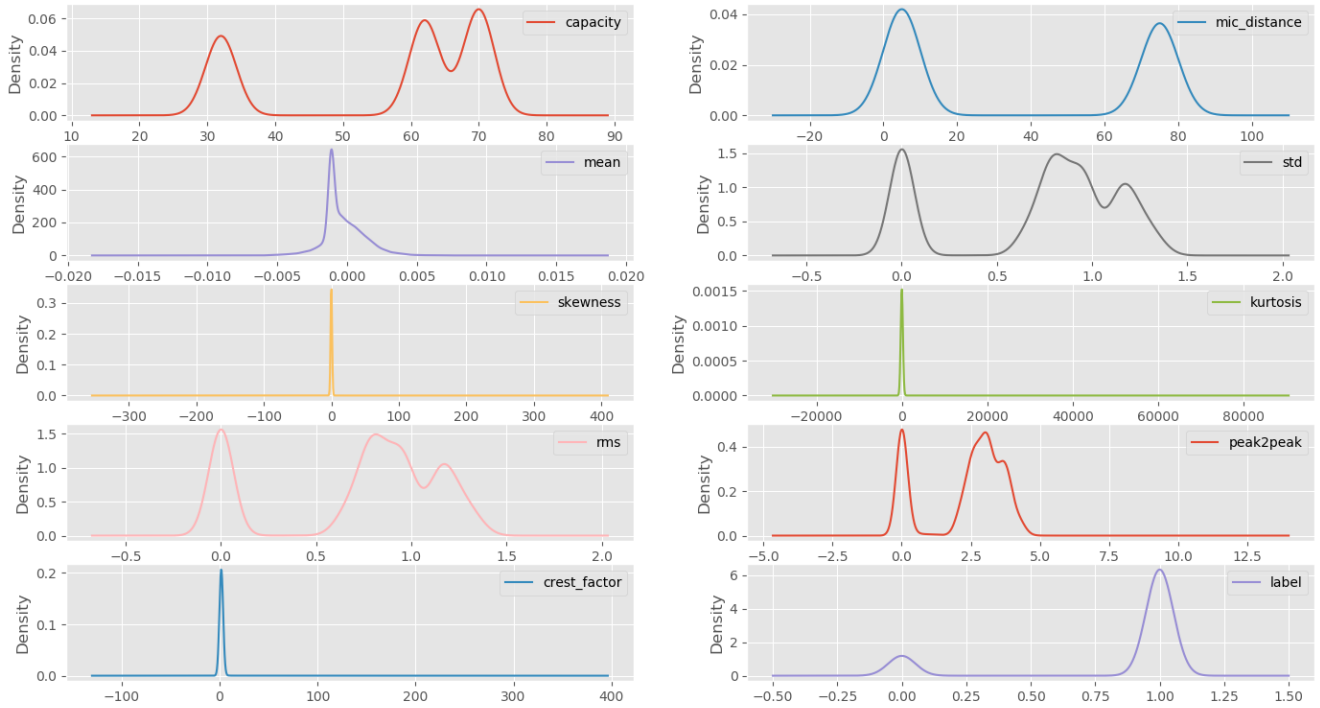


Figure 4.2.: Kernel density estimations for the different features in the data-set

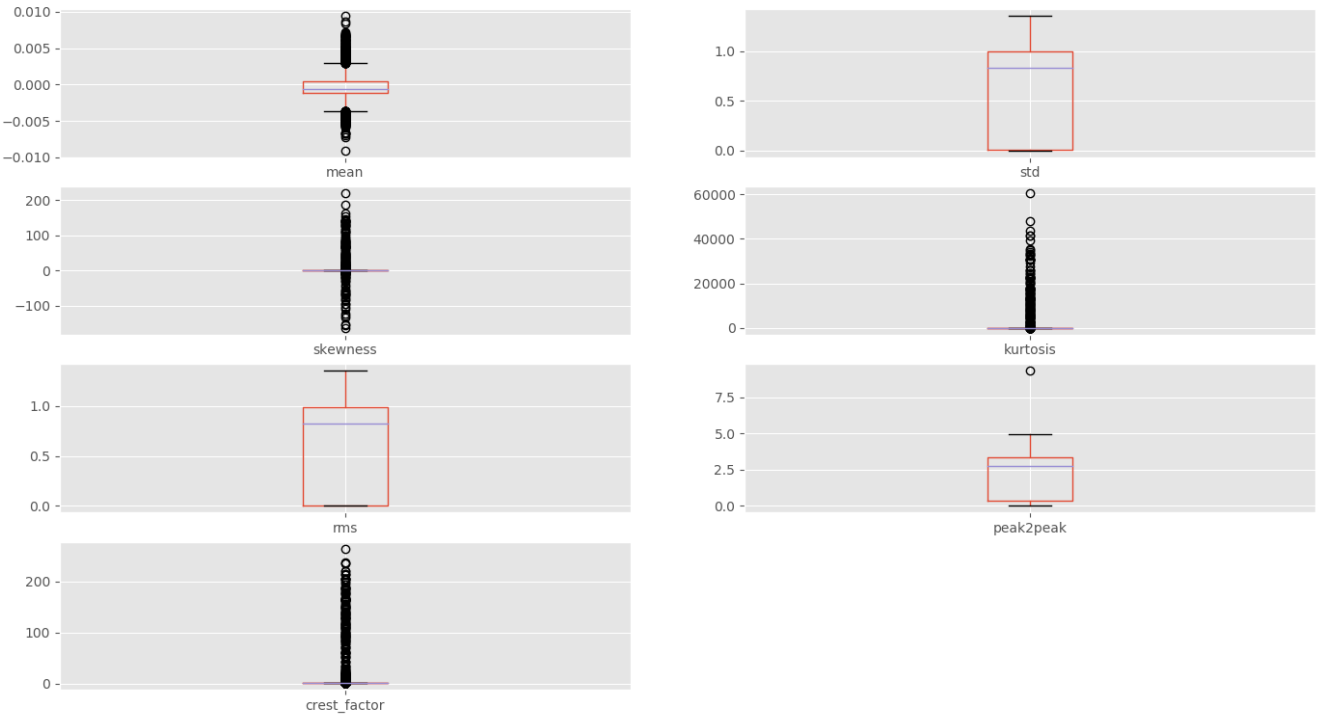


Figure 4.3.: Median value and outliers for the numerical features in the data-set

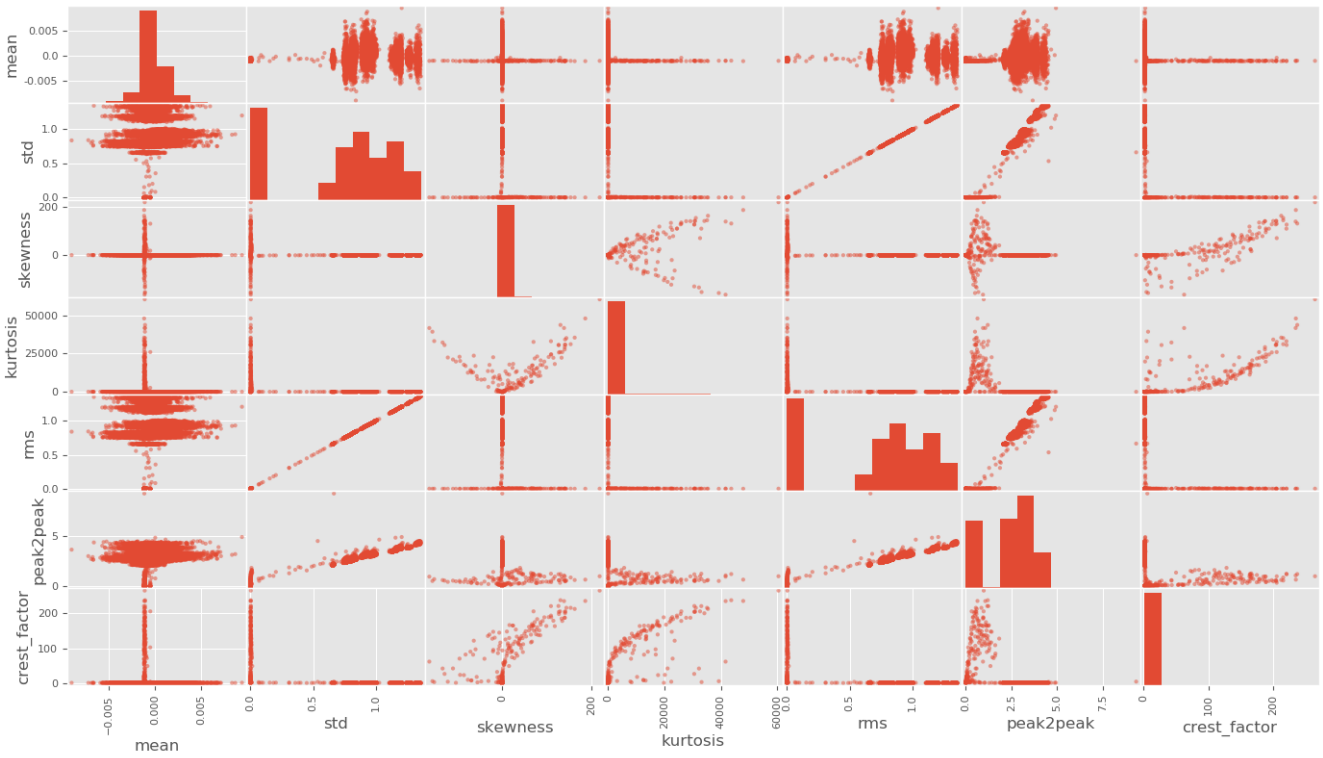


Figure 4.4.: Scatterplot matrix for the different numerical features in the data-set

tion of the two distinct values with majority of the tank capacity consisting of 62 liters and 70 liters, while a lower number of tanks falling under the 32 liters capacity. We can also see that our data is heavily imbalanced, with a vast majority of the class labels falling under the "leaked" category. In the next chapter we will learn more about this aspect of our data-set, and find out ways to achieve a solution.

Next we move on to the continuous variables in the data-set. We can see a perfect Gaussian (normal) distribution for the features kurtosis, skewness and crest factor, while the mean follows a somewhat exponential distribution. The standard deviation and root mean square appear to have the same characteristics. This is somehow anticipated, since for mean close to 0 these two can be considered identical, and makes any one of them redundant for our learning model. Let us take a closer look into only the numerical univariates in our feature list. We use the box plot for visualizing, which essentially displays the summary statistics in terms of minimum, first quartile, median, third quartile, and maximum. Figure 4.3 shows the median values and interquartile ranges (IQR) for each feature in the data. We see large number of outliers for kurtosis, skewness and crest factor, which is expected for a normal distribution. Outliers are considered to be the points that are either at least $3 \times IQR$ above the third quartile or at least $3 \times IQR$ below the first quartile. However, for skewness the outliers seem to be evenly distributed over its median value. The remaining attributes appears to be free of outliers.

The next part consists of finding correlations between different features. The correlation matrix helps us to understand the relations between two variables. Directly proportional variables are said to be positively correlated, while inversely proportional ones a termed negatively correlated. Another useful metric for finding out relations between variables is the scatter plot,

which essentially plots all the variables against each other. This can give us an idea about how linearly separable two variables are, and the symmetric or asymmetric nature of those relations. For example, Figure 4.4 shows the relation between the numerical features in our data-set.

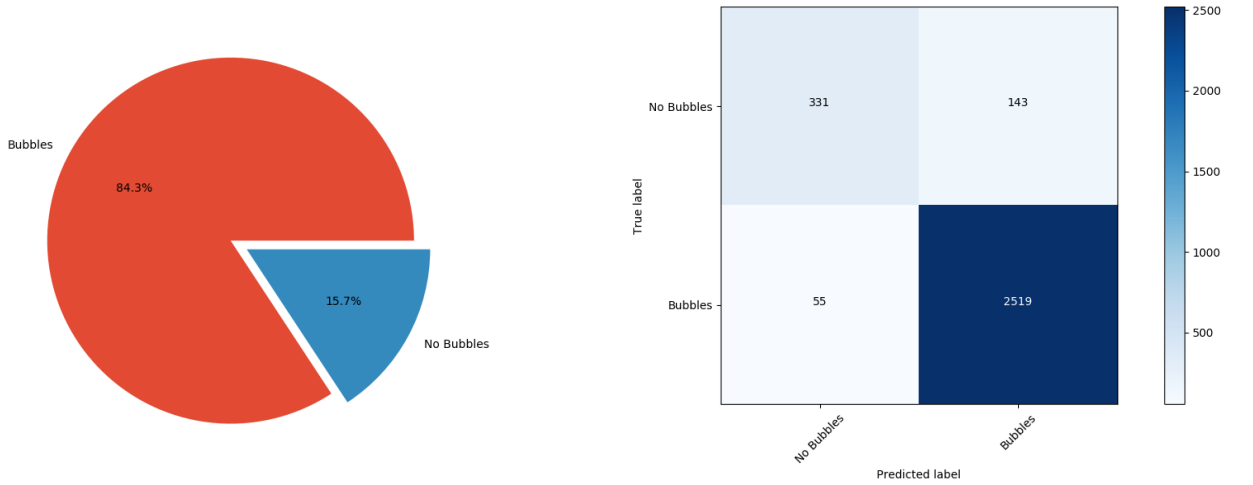
Only a handful of features (e.g. mean vs std) can be linearly separable, while most are inseparable on the linear plane. We can draw good knowledge from this when deciding on our choice of classifiers at the very beginning, even before we obtain the results and apply optimization techniques like parameter tuning. Apart from the ones mentioned above, there are other similar visualization techniques to summarize the data characteristics.

4.3 Imbalanced Class Problem

In the last segment we got an impression that the number of leak instances in our training set far exceeds the number of normal instances. This is a classic case of an imbalanced class, and often an undesirable characteristic of real world data instances. We introduced the concept in Chapter 2.2, and in this section will find out possible solutions or workarounds. This can be seen in Figure 4.5a, which shows the class distribution of 1 sec batches for our entire experiment data. We can clearly see the data is skewed towards positive instances of bubbles. Going back to the classification of imbalanced classes introduced earlier, we can categorize this particular imbalance as extrinsic in nature. We can argue that it is highly unlikely that majority of the tanks are faulty to begin with. On the contrary, it is more likely to be the opposite. However, selective data collection at a particular time or for particular reservoirs can be attributed to the nature of the imbalance we observe.

As discussed earlier, training on this imbalance class can give misleading results. We get a prediction accuracy of 0.93 for detecting bubbles. This gives the impression that our classifier is highly optimized and is predicting with a very high accuracy. Now, because of the skewed nature of the data, a random guess will have a prediction accuracy of 84%. So in reality we only get a 9% improvement in our classifier performance. The ROC-AUC curve, discussed in Chapter 2.3 is a more accurate metric when dealing with class imbalances. For the above data-set we obtain an AUC score of 0.85, which is at par with random guessing, and much lower than the obtained accuracy score. The shortcomings when dealing with imbalanced classes can also be observed from the confusion matrix. Our classifier provides a varied degree of accuracy for the two classes. The majority class 1 has a very high degree of accuracy as seen from Table 4.1. On the other hand, the metrics for class 0 give a very different picture. A recall of 0.70 means a high number of false negatives, and the overall f1-score, that combines both the precision and recall into a single metric, is much lower when compared to the majority class. We need to design a classifier that provides high (and similar) degree of accuracy for both majority and minority classes.

We chose random undersampling (Chapter 2.2) of the majority class as our sampling method. One reason is to keep the sampling process simple and to use minimum resources. We have to keep in mind that the sampling will be done at the edge, which is a low powered prototype PC or can even be a micro-controller. Random sampling has its own drawbacks. Removing chunks of data randomly can degrade the classifier performance. Yet, we can argue that since



(a) Class distribution for bubbled and non-bubbled data instances in the input data-set. We can see bubbled data instances consist bulk of our input data.

(b) Confusion Matrix for the current imbalanced data-set. We can see a large number of true negatives for prediction using the Random Forest classifier.

Figure 4.5.: The class imbalance representation for our data

	Precision	Recall	F1-score
0	0.86	0.70	0.77
1	0.95	0.98	0.96

Table 4.1.: The classification report for the unbalanced data

the data is collected originally at 100 KHz, we will have sufficient data after re-sampling to preserve all the characteristics of the majority class. However, in the later stages we are implementing signal quantization methods to reduce the bitrate of the original signal. We will see how reducing the majority class on the quantized data-set affects the classifier performance. Since we are using probabilistic methods to improve our prediction accuracy, we can allow degradation of our classifier performance up to a certain level if it means better utilization of hardware resources. However, the important thing is to have quantitative checks and measures to track the effectiveness of all these optimizations, which we will carry out at a later stage.

4.4 Signal Quantization

The data acquisition systems currently used on field has a sampling rate of 100 KHz, and we can argue that a lower sampling can also be sufficient for our use case. We would like to re-sample the data mainly for two reasons; 1. To have the option in the future to replace the existing DAQs with lower sampling rate (hence low cost) alternatives without any degradation of our classifier performance, and 2. To check if it helps reduce the processing and storage at the edge device, where we are executing the prediction algorithms. The latter might be counterproductive, since the resampling process itself will consume some resources. We conducted some benchmark tests to check the resource usage for both sampled and non-sampled data, shown in Chapter

7.3. However, irrespective of those results, it is still essential to have the option to resample the data based on the requirements. It is much easier and cost effective to scale up a single gateway if it gives the flexibility to replace the large number of current sensors with cost effective options.

According to the Nyquist-Shannon theorem if we have a signal that is perfectly band limited to a bandwidth F, then we can collect all the information there is in that signal by sampling it at discrete times, with the sampling rate being greater than $2F$ [53]. However, no real world signal, including ours, is perfectly band limited. So in other words, no data acquisition system can sample real world data perfectly. We need to be careful in deciding the resampling frequency for our data. Being falsely optimistic and choosing a value close to the Nyquist frequency may result in erroneous data reconstruction, while selecting a fairly high value being falsely pessimistic will beat the purpose of optimizing resource usage and reducing the need for expensive hardware in the first place. A trade-off between the two needs to be considered, while also taking into account that sensors and the audio signals collected may vary depending on the environmental and location changes.

We use the prediction score of our classifier on the original 100 KHz sampled data to be the basis of the evaluation for the resampled data. If after resampling we achieve a score that is similar to the former, we can conclude that our reconstructed data manages to preserve all the important characteristics of its original. We can, however, argue that since we're using statistical methods to improve our prediction results, a lower accuracy score can be accepted as well.

For the quantization, we apply a method commonly used in production engineering. We begin by taking a single random specimen of 1 sec data, for both files labeled as bubbles and without bubbles from the original data-set. We start by transforming the signal from the time domain to the frequency domain. Next we determine the frequency of the signal at the $-3dB$ point (cut-off frequency). This is also known as half attenuated or half power level frequency, and comes from the ratio between output power and input power, expressed as:

$$P_{dB} = 10 \log_{10} \left(\frac{P_{out}}{P_{in}} \right) \tag{4.9}$$

Since the output power is half of the input power, we get the value $10 \log_{10}(0.5) = -3.01\, dB$. The Nyquist-Shannon frequency that we obtain from this step is used for the resampling of our entire data-set. Figure 4.6 shows the cutoff frequency to be at 50 Hz, which means the resampled frequency should be above 100 Hz. The selection of the exact value, however is more of a design choice, and can vary depending on a multitude of factors, even on the field of engineering it is applied to. We chose this to be 10 times our obtained cutoff, i.e. at 500 Hz. This is not purely a random choice. It is desirable to have a 'buffer' region over the minimum requirement of 100 Hz to accommodate the changes in signal characteristics that may occur over time. For the resampling, Fourier analysis is used, which is a process of denoting a signal as a sum of its periodic components and reconstructing the signal from those components. We use Fast Fourier Transform (FFT), which is a speedy algorithm to compute the Discrete Fourier Transform (DFT). The DFT is a specialized version of the Fourier transform that deals with discretized signals. The FFT $y[k]$ of length N on the length-N sequence $x[n]$ is denoted by the following equation:

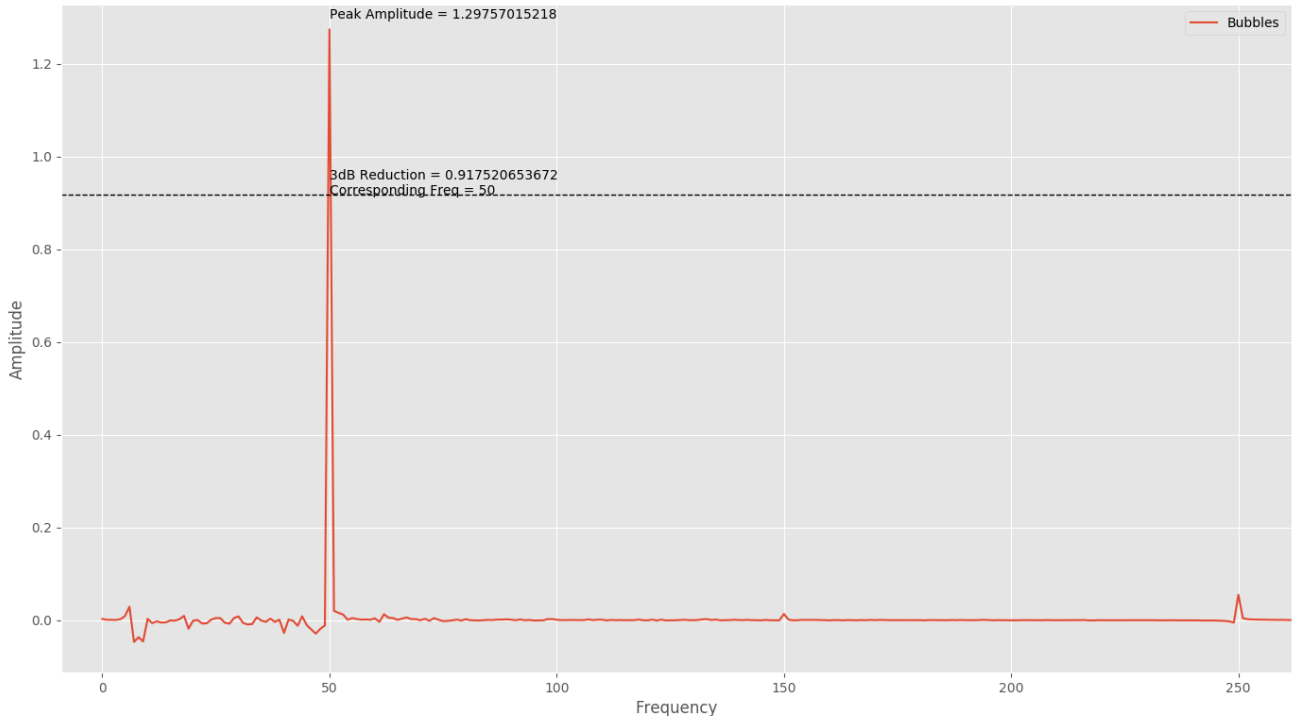


Figure 4.6.: The -3dB cutoff frequency for both bubbled and non-bubbled data specimen, which corresponds to 50 Hz.

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n] \tag{4.10}$$

Almost all modern languages provide fast and effective signal processing libraries for the above tasks. We take advantage of a similar module [26], which uses Fourier transformation to re-sample data along a given axis.

To use a sampled-time signal in a continuous-time system it must be converted back into a continuous-time signal. This transformation from sampled to continuous time is called signal reconstruction. Reconstruction is done by the interpolation of the output signal, where the value of the continuous-time signals between samples is constructed from the previous values of the discrete-time signal. We see in Figure 4.7 that our interpolated signal closely follows the original signal with minimal aliasing. This can be further verified using the classification algorithms. Ideally it is expected to give similar prediction scores for both the original and re-sampled datasets, and indeed exhibits identical accuracy levels in our experiment. This will be discussed further in Chapter 5.3 (model training).

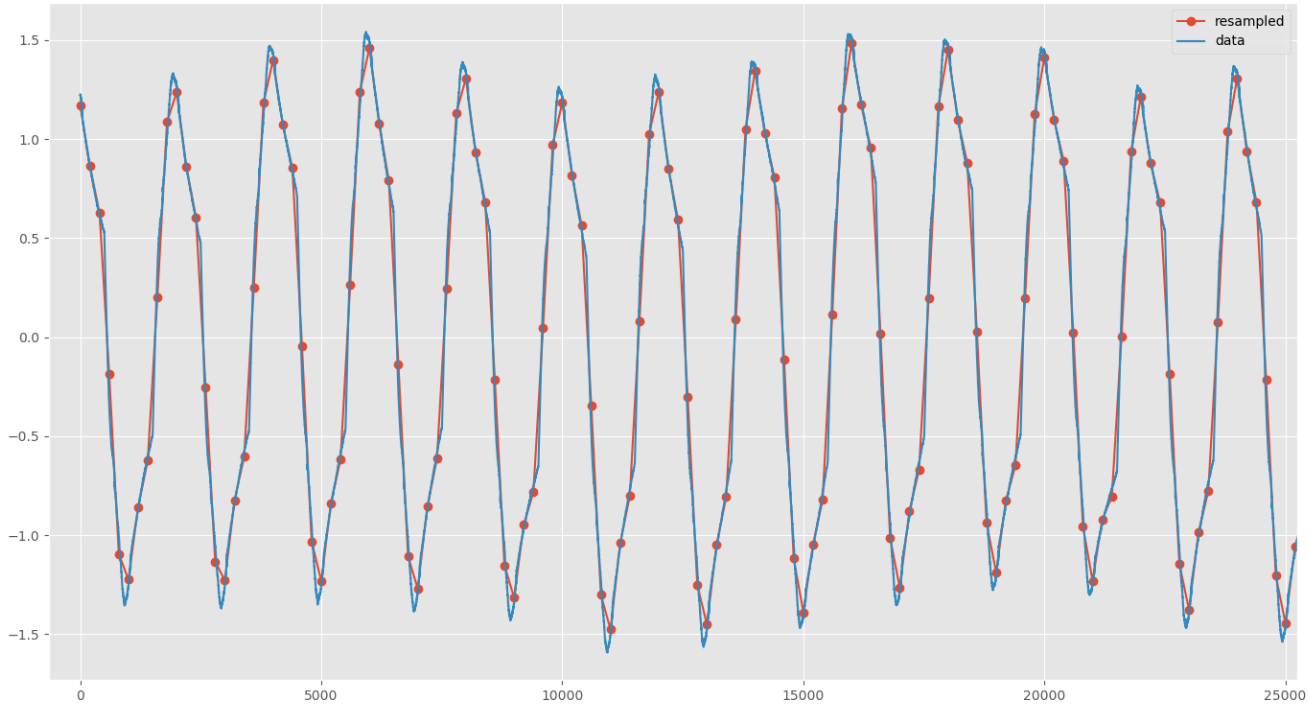


Figure 4.7.: The interpolated signal (red) and the original signal (blue). We can see minimal aliasing.

5 Model Training

Here we introduce the concept of a baseline classifier and provide a detailed comparison of our classification models over that baseline.

5.1 Baseline Classifier

In the first chapter we introduced some classification models that will be used for training our data. But before moving into the evaluation of those models, we need to define a baseline performance that will act as a benchmark for those. This can be easily achieved by applying a baseline classifier (also called a zero-rule or dummy classifier) to the data-set. In the next paragraphs we explain the working of a baseline classifier very briefly, before setting up the minimum benchmarks (the performance floor) that our models need to improve upon.

A baseline classifier provides a point of comparison for the evaluation of the more advanced classifiers, and brings a quantitative measure as to how much better the applied model is. Though similar in its purpose, baseline classifiers can apply different algorithms to emulate a real classifier. Two of its most commonly used algorithms are the random guess and zero rule algorithm. The random guessing method, in its simplest form, predicts a random outcome from the training data-set. The probability of picking up a single outcome will depend on the distribution of classes. For example, a balanced class of two distinct outcomes will have a scoring accuracy of around 50%. On the other hand, a zero-rule algorithm only depends on the target values and does not take the features into consideration. It simply predicts the most occurring outcome from the training data-set. Again, for a balanced class with two outcomes, it will return a prediction accuracy of 50% for both classes. However, for imbalanced classes the zero-rule is considered a better baseline than random guessing since it eliminates the element of 'chance' associated with the later.

Before implementing the baseline classifier, let's look back in Chapter 4.3 where we took care of the imbalanced class problem by downsampling the majority class to be roughly equal to the other class. This would mean our baseline model should have a theoretical prediction accuracy of about 50%. However, we can apply the baseline to both the balanced and unbalanced data-set to examine if it correctly captures the distribution of both instances. The dummy classifier we have used [49] has diverse strategies for random guessing like stratified (employs the class distribution of the training data-set), most frequent, and random sampling based on uniform class distribution, or even lets the user specify a certain class. We used the most-frequent strategy to make it work as a zero rule based classifier. The Table 5.1 shows the confusion matrix for both balanced and imbalanced data-set, for which we again refer the findings of Chapter 4.3 that shows the occurrence of bubbles (class 1) is at around 84% of the total data.

The results seen are in line with our theoretical assumptions and represents the class distributions accurately for both the data-sets. The ROC-AUC curve (Figure 5.1) for both data-sets would give the same area of 50%, since it does not take class imbalances into consideration. On the other hand the accuracy scores of the model is heavily dependent on the class distribution.

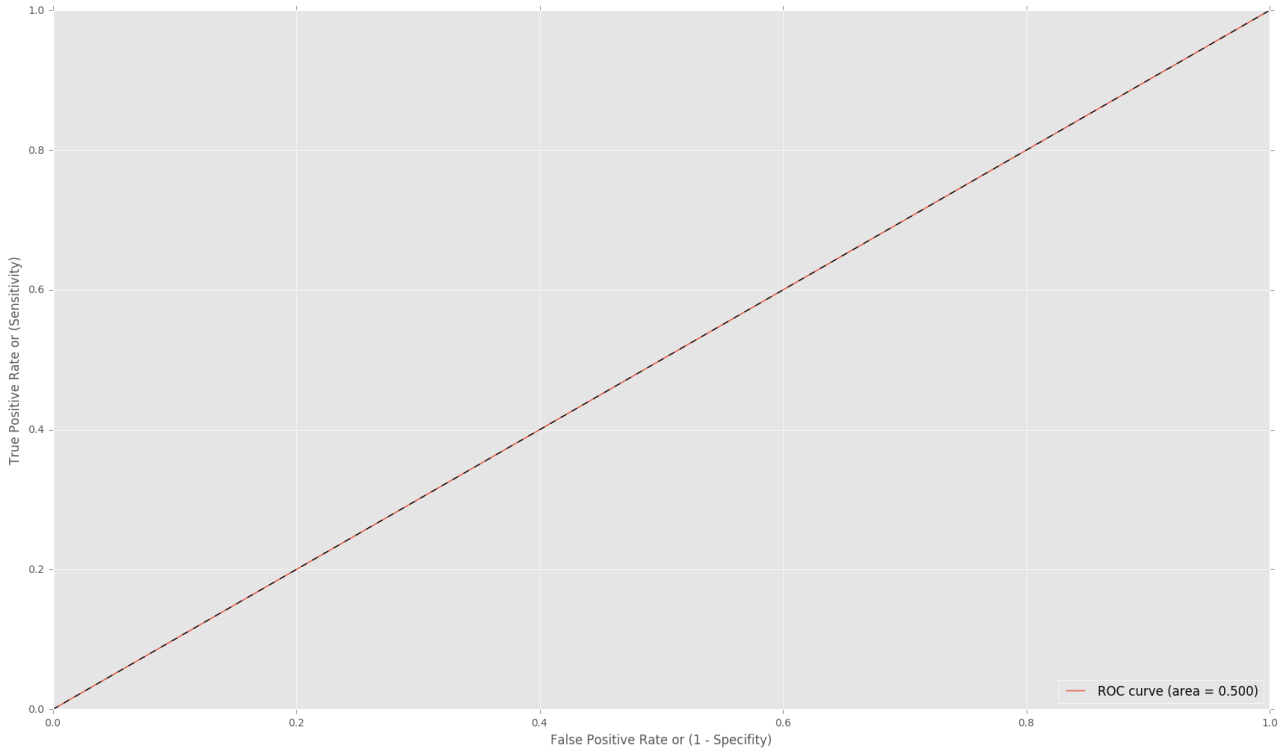


Figure 5.1.: The ROC-AUC curve for the baseline classifier. We can see that the baseline performance (red line) is exactly the same as random guessing (dotted line).

Unbalanced Data

	Predicted	
True	No Bubbles	Bubbles
No Bubbles	0	496
Bubbles	0	2552

Balanced Data

	Predicted	
True	No Bubbles	Bubbles
No Bubbles	0	465
Bubbles	0	503

Table 5.1.: The confusion matrix obtained from the Baseline Classifier for both the unbalanced (left) and balanced (right) data-set.

However, this is trivial in our case, since we balance our data-set beforehand which allows us to use the accuracy score and ROC somewhat interchangeably.

5.2 Classifiers Recap

Now that we have established our baseline for comparison, we can proceed with implementing our classification models and evaluate how they perform. We refer to Chapter 2 for a brief introduction to different learning methods and their architecture. We decided to use three distinct methods to train our model. This is partly as to provide us with a comparative analysis for our probabilistic models in the later chapters. Another reason is to add greater flexibility to the process. Not all classifiers perform equally or demand the same resources. There should be in essence an option to switch back and forth between different learners with relative ease depending on the hardware and use case. We can have three separate trained models for the same

data-set that can be loaded for prediction on the desired hardware making it less dependent on the data collection setup. Moreover, though certain classifiers tend to adapt well to all kinds of data, the performance generally is dependent on the characteristics of the data-set. Our model should be future proof to take care of the changes that might be encountered in the future in terms of the equipment, environment or user requirements. We begin with a very short recap of the models we have used for training our input data, before moving to the evaluation results and the comparative analysis of those models.

Random Forest (RF)
A type of ensemble learning, which employs a collection of decision trees derived from randomized subsets of the training data. It uses the bagging method in which each tree is created independently with samples from the training data. Finally a majority vote is taken among all the trees to get the final prediction. This differs from the standard decision tree algorithms where the best fit is decided based on the entire training data and not on randomized subsets, thus reducing the variance.

Support Vector Machines (SVM)
A supervised learning method frequently used in classification problems and focuses on finding the best separation plane for the two classes. The idea is to maximize the distance (called margin) between the closest data points on either side of the decision boundary, while being rigid to outliers in the data. For values that are not linearly separable, it employs a method known as the kernel trick, which is basically a set of functions (depending on the problem) that transforms a lower dimensional data space into higher dimensions. For our data, we use the Radial Basis Function (RBF) [8] with coefficients of 1.0, obtained via Grid Search CV.

Extreme Gradient Boosting (XGBoost)
An enhancement to the tree boosting model offering a more robust, scalable and optimized performance. It uses a slightly modified version of the greedy algorithm known as the approximate algorithm [10] for better scalability and resource consumption. Moreover, the algorithm is optimized to take advantage of modern computing architectures like clusters and large scale parallel processors.

Before moving forward to the next section, we briefly mention the strategy used to partition the available data into train, test and validation sets. Creating a partition of our data-set for testing that is kept separate at all times from the training data provide us with an honest evaluation of our predictive models. For the train-test split we use a common approach to allocate 20% of the data for testing, while the remaining 80% is used to train the model. For the validation set, Stratified k-fold method is employed on the training data-set which is further utilized in parameter tuning.

5.3 Model Comparison

Next, let us move forward to implementing these models for our data. Our first goal is to obtain the hyper-parameters for maximum optimization, which is easily achieved with the application of Grid Search CV. For the next part we train the model on our data-set using the freshly achieved

tuning parameters. Finally, we evaluate the prediction accuracy for our model in terms of the metrics introduced in Chapter 2.3. The first two are more of a programming construct and will vary with the implementation of different ML languages and libraries, so we focus mainly on the evaluation metrics of the models, but not before introducing the important concept of feature standardization.

For the ensemble learners we employ the training set can be used without any adjustment, however we normalize [16] the data for fitting the SVM model. Normalization (sometimes used synonymously to standardization or feature scaling) is a useful trick in machine learning to homogenize the values of the different features. An uncomplicated way of achieving this is to rescale the features in the range [0 to 1] or [-1 to 1]. However, a common method widely used in ML algorithms including SVM is to transform the individual features so that they have zero mean, expressed by the formula.

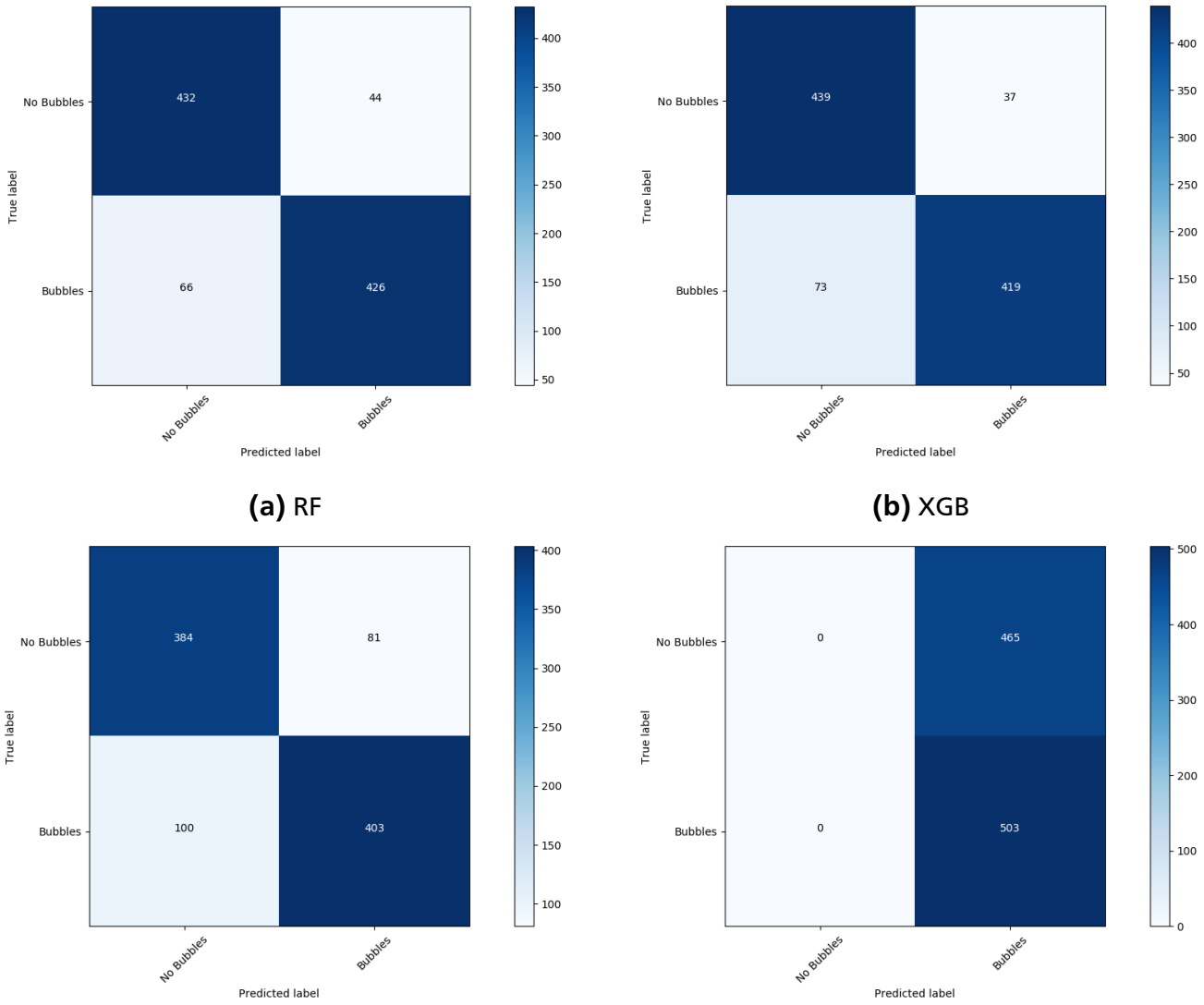
$$\bar{x} = \frac{x - \bar{x}}{\sigma} \tag{5.1}$$

where $\bar{x}$ and σ are the mean and standard deviation of x respectively.

Standardization is an important pre-processing task for SVM. The decision boundary is calculated based on the distance between the linear separating plane and the classes nearest to the boundary (support vectors). If one of the features in the training set has very high values compared to others, that feature will have a much higher influence in calculating the boundary. This makes it essential to convert all the features into similar reference points in terms of their value ranges.

Finally we can proceed with the performance of our classifiers. Let us start with the confusion matrices for the different learners depicted in Figure 5.2, followed by the individual metrics shown in Table 5.2. A binary classifier can only make two types of errors, the false positives and the false negatives, which can be easily visualized with the help of the CM. A quick glance and we can tell that there is a higher accuracy of correct classifications for both RF and XGBoost when compared to SVM, which shows a higher degree of false negatives. This is in line with our expectations, since an ensemble of learners tend to perform well for a majority of datasets as compared to a single learning algorithm. However there is still room for improvement, particularly for the number of false negatives which can have catastrophic implication for our use case since we do not want faulty tanks to get undetected. False positives will simply raise false alarms, which though inconvenient, can be quickly mitigated by cross checks. We take a different approach to tackle this, instead of treating it as our final model we use our classifier's performance measures as an input for a probabilistic method to consolidate on the predictions. We will discuss this in the later chapters.

Similar results can be obtained from the ROC curve for the different classifiers (Figure 5.3), which shows the overall accuracy to be close to 90% for both RF and XGB, a considerable improvement over the baseline score. The SVM performs slightly worse than the other two, but yet has its own advantage as we will see in Chapter 7.2. Two of the parameters (sensitivity and specificity) in particular are of importance to us and will be carried forward to assist with our prediction improvement tactics.



(a) RF **(b)** XGB

(c) SVM **(d)** Baseline

Figure 5.2.: The confusion matrix for the different classifiers.

	RF	SVM	XGB
Accuracy	0.889	0.813	0.886
Precision	0.906	0.832	0.918
Recall (Sensitivity)	0.867	0.801	0.851
Specificity	0.909	0.825	0.922
F1-Score	0.890	0.820	0.880

Table 5.2.: Accuracy metrics for the different classifiers.

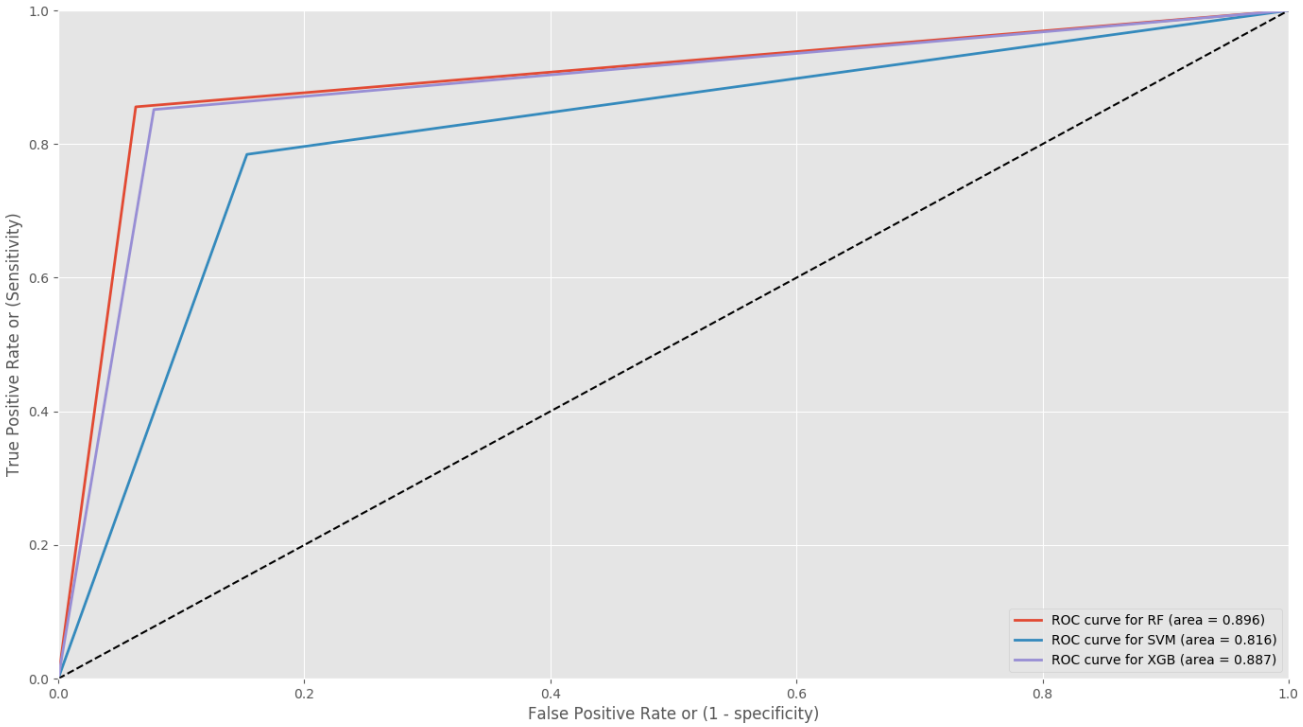


Figure 5.3.: The ROC-AUC curve for the three different classifiers. We can see that both RF and XGB models have similar accuracy.

Before concluding this chapter, we will examine the effect of signal quantization on the prediction accuracy. This is in continuation to the previous Chapter 4.4 where we argued that the reconstructed (lower bitrate) signal manages to preserve all the important characteristics of its original. The best way to confirm this would be to compare the prediction accuracy of both the signals. Figure 5.4 shows the ROC-AUC for both the original and quantized signal for the RF classifier. We can indeed observe that both have a very similar prediction accuracy, thus reinforcing our previous observations.

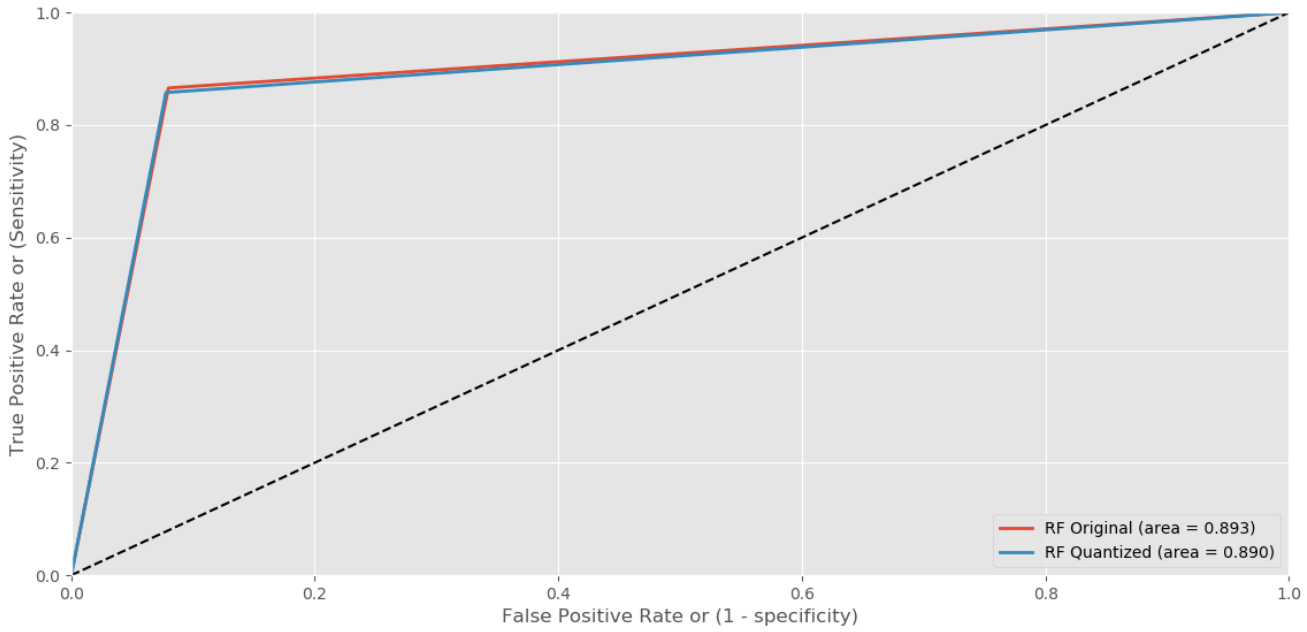


Figure 5.4.: The ROC-AUC curve for the RF classifier of both the original 100 KHz and resampled 500 Hz data. We can see that both of them exhibit similar accuracy.

6 Bayesian Inference

In this chapter, we apply the concept of Bayesian Inference on the predicted outcomes to further improve the accuracy and present the results through different examples involving both hypothetical and real data from our experiment.

6.1 Methodology

In Chapter 2.9 we discussed how statistical inference methods, particularly ones applying the Bayes' theorem can be used to improve the predictive models, and the motivation behind it. In this chapter we will build a BI model that relates to our prediction results, check how it performs and to what extent it improves our classifier performance. We can choose any of the three classifiers from Chapter 5.3 as the foundation for our probabilistic model, and it is worth noting that BI methods complement a classifier performance by borrowing some of its input parameters from the classifier itself. We choose our classifier as Random Forest for implementing the Bayesian inference.

Before diving into the implementation of our probabilistic model, we would like to introduce the concept of distribution, which is basically a set of values and their subsequent probabilities. For discrete probabilities that only assumes certain values, we can express the aforementioned distribution as the probability mass function (PMF). More explicitly, for a probability distribution that can only assume two distinct values (like leak or no leak for our use case), it is said to be a Bernoulli distribution. It takes one parameter p which can be seen as the probability of getting a positive result, and returns 1 for that outcome or 0 otherwise for $1-p$ (for two equally likely outcomes p takes the value of 0.5). For a random input variable X, the Bernoulli distribution equation can be written as:

$$P(X=1|p)=p \text{ and } P(X=0|p)=1-p \tag{6.1}$$

for both the probability of getting a positive and negative detection respectively. The two expression can be combined into a single equation:

$$P(X=x|p)=p^x(1-p)^{1-x}, \quad where\ x=(0,1) \tag{6.2}$$

The above equation is essentially the PMF for the distribution. It is worth mentioning that the Bernoulli distribution is a specific case where only a single trial is considered. A more generalized distribution that takes into account several independent trials is known as the Binomial distribution. We can easily expand the above equation to multiple independent trials, expressed as:

$$P(k|p)=p^{\#positive\_detection}(1-p)^{\#negative\_detection} \tag{6.3}$$

However, there is a small caveat to this. A binomial distribution is only applicable to a set of independent trials, where each outcome is not dependent on the previous one. However, in our case each outcome is interdependent to their posterior probabilities, and as such cannot be

represented by the equation 6.3 in its current form. Many propositions have been made to generalize the binomial distribution to represent dependent trials [2][40]. We will not go into details as it might deviate the current focus. It is however, important to note that with generalizations of the above principles, it is possible to create realistic models for our use case with relative ease.

The first step of BI is to build a model that precisely represent our given data-set. We can use the above equation as the probabilistic model for our data, with the PMF being the likelihood function. In the next section, we will generate the prior and posterior distributions for our data using this PMF, and check how it changes with new information.

Once we have established the probabilistic model, the next step is to create the algorithm for our Bayesian learning. To achieve this we take the help of the Bayes' theorem introduced in Chapter 2.9. In our case we have just two probabilities, the experiment samples indicating bubbles in the tank or not. The Bayes' theorem for our model can be rewritten as:

$$P(B|+) = \frac{P(B) \times P(+|B)}{P(+)} \tag{6.4}$$

$$P(NB|-) = \frac{P(NB) \times P(-|NB)}{P(-)} \tag{6.5}$$

where,
$(+)$ and $(-)$ indicate positive and negative detections respectively.

$P(B)$ and $P(NB)$ are the prior probabilities for bubbles and no bubbles respectively.

$P(B|+)$ is the conditional probability of bubbles given positive detection while $P(NB|-)$ is the conditional probability of no bubbles given negative detection.

$P(+|B)$ and $P(-|NB)$ are the conditional probabilities of a positive (negative) detection given bubbles (no bubbles).

Let us decode each of these in a bit more detail. For the prior probabilities $P(B)$ and $P(NB)$, we have unfortunately no background data to set our assumptions. For this reason we have assigned equal prior probabilities of getting bubbles or no bubbles at the initial phase before we see any new instance. This is unlikely for real world data since only a small fraction of the tanks will have leaks. However, we must remember that our data was heavily skewed towards faulty instances, and down-sampling is employed to create an equal distribution of the two classes.

$P(B|+)$ and $P(NB|-)$ are essentially the tenets for our performance improvement. If there is a positive detection, we would like to know how probable that there is indeed bubbles in the tank. Conversely, on a negative detection we would like to ensure that the tank is indeed reliable. These values will be constantly updated with each new data instance (recap: for each specimen we have 120 data instances).

$P(+|B)$ and $P(-|NB)$ are the known parameters in our equation. $P(+|B)$ can be rephrased as the number of detections that are correctly classified as bubbles, which is identical to the

definition of TPR (or sensitivity or recall) for a classifier, and can be obtained from classifier evaluation. Similarly, $P(-|NB)$ can be expressed as the number of detections that are correctly classified as no bubbles, and in essence the same as the TNR or the specificity of the classifier. The sensitivity and specificity values of the different classifiers can be taken directly from Table 5.2 obtained during the evaluation of our models. This reiterates our claim that BI does not necessarily need to work in isolation, and can make use of the classifier output to further enhance model performance. The joint probabilities of bubbles and detected or no bubbles and not detected can be expressed as:

$$P(B\,and\,+) = P(B) \times P(+|B) \quad and \quad P(NB\,and\,-) = P(NB) \times P(-|B) \tag{6.6}$$

6.2 Application

Now that we have obtained the prior probabilities and the likelihood function for our model, we can go ahead and apply the BI framework. Figure 6.1 illustrates the algorithm for the model. We can best explain the model with the help of a few examples. Let us suppose we have an input stream of two consecutive 1s, which are basically the predicted outcomes from the classifier. Before seeing any data the prior probabilities are set to 0.5, i.e. the chances of seeing bubbles and no bubbles are equally likely. Now when a new data instance goes through the classifier, there are two competing claims, the batch contains bubbles or does not, represented by the top branch of our probability tree. Since the instance is 1, we know it is a positive detection, so we are only concerned with the first and third branch from the left. Given the prediction is positive, we would like to know the probability that there is indeed bubbles, or $P(B|+)$, for which we have already established a formula given in equation 6.4. The numerator can be obtained by multiplying the sensitivity of the classifier with the prior, while the denominator $P(+)$ can be obtained from the sum of the joint probabilities $P(B\,and\,+)$ and $P(NB\,and\,+)$ as both are disjoint outcomes. The result obtained is our posterior probability of bubbles for this iteration, which comes at around 0.90 or 90%. For the specific values used in the calculation we refer to the tree in Figure 6.1, which we substitute in the Bayes' theorem introduced earlier.

Our model has learned on the new data, and has updated the probability of bubbles from 50% to 90%. Let's see what happens if we have another positive detection. For this iteration, our prior values has been updated to $P(B) = 0.90$ and $P(NB) = 1 - P(B) = 0.10$ based on the previous results. By repeating the previous step, we arrive at a new posterior probability of 98%, an increase from the previous value. Now if it encounters a negative detection next, it will travel through the $2nd$ and $4th$ branch of our probability tree and arrive at a lower posterior probability for bubbles than its previous step. This idea of constantly learning with new data to update the probability of an outcome is the essence of Bayesian learning. Let us consider another example that includes new instances for both bubbles and no bubble data. Table 6.1 shows the step by step use of the algorithm for a slightly bigger input stream of $[1, 0, 0, 1, 1]$. The algorithm is based on the general Bayes' theorem, and can extend to any given length of the input array. The individual values are derived from equations 5.4 and 5.5. As mentioned earlier in Chapter 3.2, each of our 127 collected specimens have 120 batches. We obtain the overall prediction for each sample by combining its constituent batches, which would indicate 120 iterations to determine the final probability. This will be described in Chapter 7.2 where we

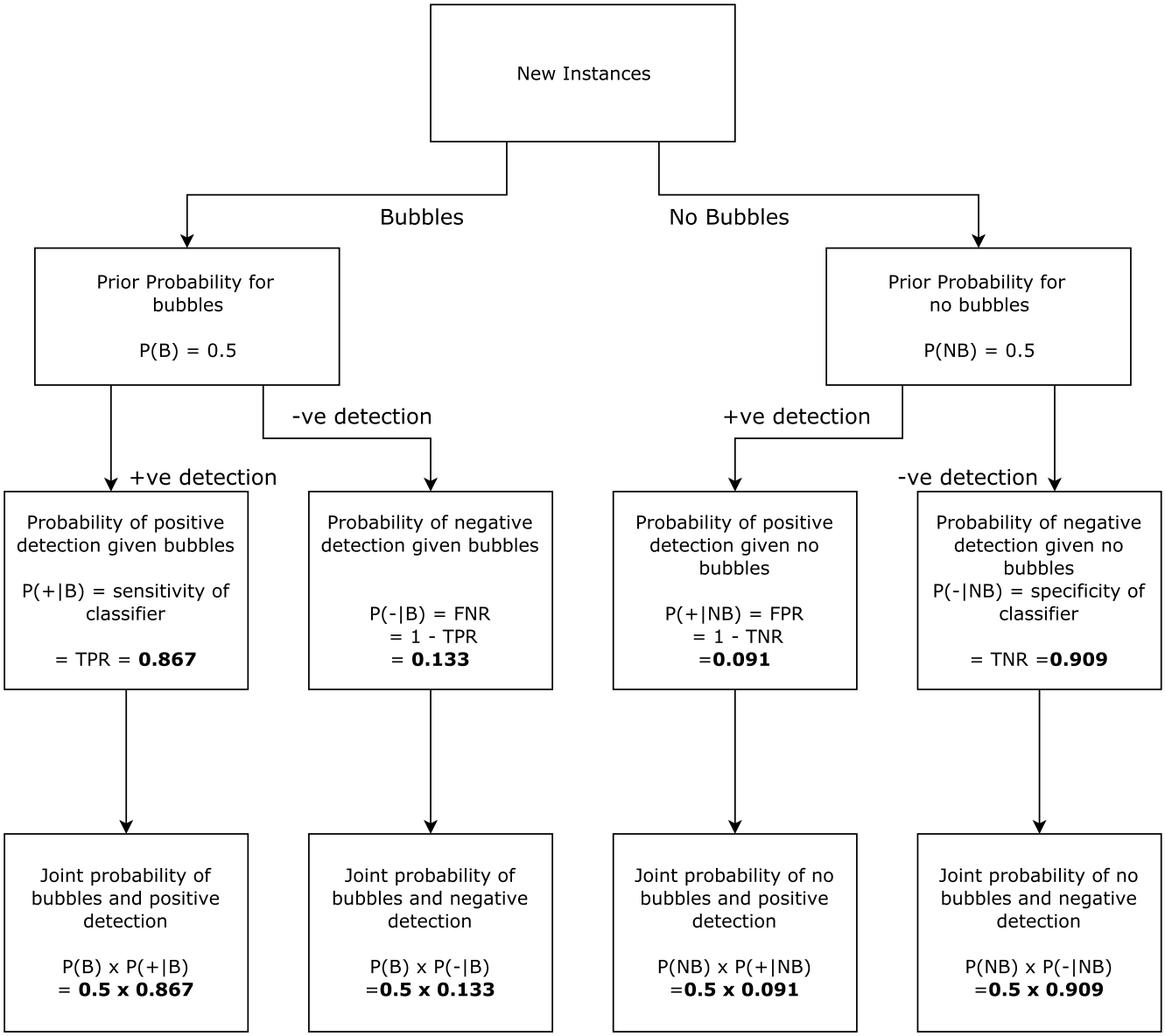


Figure 6.1.: The decision tree for the Bayesian Inference algorithm.

	Prior		Joint probabilities				
Input	P(B)	P(NB)	P(B and +)	P(B and -)	P(NB and +)	P(NB and -)	Posterior
1	0.500	0.500	0.4335	-	0.0455	-	0.905
0	0.905	0.095	-	0.1203	-	0.0863	0.582
0	0.582	0.418	-	0.0774	-	0.3799	0.169
1	0.169	0.831	0.1465	-	0.0756	-	0.659
1	0.659	0.341	0.5713	-	0.0310	-	0.948

Table 6.1.: Implementation of the Bayesian Inference algorithm for an example input stream of predicted data $[1, 0, 0, 1, 1]$

Filename	Label	Bayesian Inference	Error	Ratio	Majority Voting
Specimen 1	1	1	0	0.54	1
Specimen 2	1	1	0	0.92	1
Specimen 3	0	0	0	0.16	0
Specimen 4	0	0	0	0.35	0
Specimen 5	0	0	0	0.07	0

Table 6.2.: An small sample from our output of the BI model, along with the error checking mechanism and majority voting as a fallback.

will formulate a method to evaluate the performance of our BI model. Once again, at the risk of repeating ourselves, we would reiterate that Bayesian methods in general can be a bottleneck to computing resources depending on the size of data. It is always desirable to get the job done with minimum iterations possible. In chapter 7.1, we will compare the minimum Bayesian window for our chosen classifiers. It is worth noting that each individual classifier will have different sensitivity and specificity values, on which the BI model is dependent.

6.3 Results and Evaluation

In the previous section we observed how probabilistic models based on Bayesian methods can be utilized to improve classifier performance. They usually do not work in isolation, but complements and even reinforces what is already learned from the classification algorithms. However, it is also useful to have an error checking mechanism to inspect and if required, evolve our probabilistic methods to better suit the data and the requirements at hand. We devised a simple yet effective method to check for inaccuracies in our BI model, while at the same time being fast and unobtrusive to the whole implementation pipeline. To illustrate the concept, we present a small excerpt from the output of the BI model in Table 6.2. Unlike the model prediction which is calculated over each of 120 batches separately for a specimen, we calculate the BI over the entire specimen. The 120 individual predictions are fed to the probabilistic model sequentially to obtain the collective probability for that tank. The above table is representative of our test set, and the labels make it easier to calculate the error which can be easily expressed as:

$$Error, \quad if\ abs(Label - BayesianInference) > 0.50 \tag{6.7}$$

However, although a good method to check our test data, this cannot be applied to real life new data instances simply because they do not come with labels. A more practical approach involves obtaining the ratio of the correct predictions for a given tank. This can be easily calculated by the ratio of correct prediction over the total predictions, given by:

$$Ratio = \frac{sum(predictions)}{length(predictions)} \tag{6.8}$$

As an example we can look at the prediction sequence of $[1, 1, 0, 1, 1]$. By the above equation the ratio can be calculated as $(1 + 1 + 0 + 1 + 1)/5 = 4/5$ or 80%. This gives us two distinct advantages. First we can check our Bayesian accuracy based on the ratio. A Bayesian prediction

of 1 should correspond with the ratio ranging from 50% to 100%, as can be seen for all the entries in the output table 6.2. However, we can have another entry for which there is an error in Bayesian calculation. Such instances are extremely rare to begin with, and can be mitigated with the use of majority voting algorithm as a fallback measure. Another advantage of representing the predictions as ratios is that instead of displaying the final outcomes in terms of absolute values, we can express them as probability percentages. This can help in the diagnostics process, as tanks with higher probabilities of leak need to be attended on a priority basis than others.

The results on our data show an accuracy of over 99% for the test set (taken collectively for each specimen). This is a considerable improvement over the 89% scores we achieved through the classifiers alone. We can argue that a majority voting algorithm would have been sufficient to obtain the same results with a much simpler implementation, and perhaps less computing resources. Indeed, Hamilton-Wright et al. on their experiment [20] with electromyography (EMG) data found that Bayesian aggregation provides lesser improvement over majority voting than previously thought. They suggest this inability as not due to the lack of data, but to the distribution of the data itself. Some counter reasoning can be produced however, especially in relevance to our use case. First, majority voting will not work for a specimen that has equal predictions of bubbles and no bubbles for all its 120 batches. In such cases, albeit rare, we will have to introduce some sort of approximation. This is a non-issue for the BI model, since even for equal prediction instances the probability will be slightly more/less than 50% (owing to the sensitivity and specificity values), giving unambiguous results. Another reason is that instead of choosing one particular class and discarding the others, the BI expresses the level of confidence for all the classes. This can however, be both a merit and a drawback. While it can provide a more accurate representation in terms of probabilities, it also dilutes the majoritarian principle of traditional voting, which is often simpler to interpret. For the next reason, we refer to the findings of Zhu et al. [57]. The results of their experiments showed that for a binary classification problem having equal prior probabilities for both instances, the effectiveness of the majority voting algorithm is constrained by the TPR and FPR of the classifiers. It was found out that the average TPR should be at least 50% and the average FPR at most 50% for the proper functioning of the voting algorithm. BI, on the other hand, does not impose these limitations.

7 Optimization Techniques and Evaluations

This chapter deals with the optimization techniques to determine the best parameters for our use case in terms of accuracy, time and resource utilization.

7.1 Minimum Bayesian window for different classifiers and batch sizes

Our implementation is based on a sequential process consisting of learning, predicting and then improving on the predicted results, each dependent on the former. Optimization techniques should be employed by treating all the different steps as a single cohesive unit, and not in isolation. In this segment we will find out the optimal balance for our models in terms of performance and resource utilization. In all our tests so far the batch interval is selected at 1 sec (Chapter 3.3) for feature extraction. This is more of an intuitive design choice, fewer batches will aggregate over the data points while too many batches may lead to overfitting and increased resource consumption. In this segment we will provide a quantitative comparison between different batches of intervals 1 sec, $1/10th$ sec, $1/20th$ sec, $1/50th$ sec and $1/100th$ sec for each classifiers. This should, theoretically, also have an affect on the accuracy of our predictive model, and will have a relaying effect to the Bayesian analysis that relies heavily on the metrics of the classifier.

The Bayesian model, owing to its working methodology, should eventually achieve the desired results which we observed in previous chapters. However, it might take fewer or more iterations, which can be examined by checking the minimum Bayesian window required to achieve a certain probability for all the window sizes, for all classifier models. We look into our experiment from two different point of views, one from the perspective of the different batch sizes for the same classifier and the other from the perspective of the different classifiers given the same batch size. At the end, we will collate the two point of views to draw conclusions from our experiment. We will also point out some unexpected results that we ran into during the experiment, and will try to provide reasoning for those irregularities to the best of our knowledge.

We begin with creating different batch sizes based on the time window for the raw data, as shown in Table 7.1. It shows the total data points for all the tanks before and after the majority class reduction, as well as for the individual classes. We would like to refer to Chapter 5.3, where we have already come across the performance metrics for 1 sec batches. We follow the same process to obtain the performance results for the other batches, shown in Table 7.2.

We see an interesting pattern emerging here. With the increase of batches we can see an overall decrease in classification accuracy. But what is striking is the rate of performance degradation is not consistent for all classifiers. We can observe sharp decrease in accuracy for the ensemble learners, while SVM shows a much slower deterioration and even gradually outperforms the other classifiers. This aspect is much more difficult to explain. While we do not know the exact reason behind this, we can certainly investigate a few probable causes. Our feature set is a collection of summary statistics for n samples, where the size of n is determined by the selection of the batch interval. Different sizes of input samples should result in slightly different distribution

1 sec	0.1 sec	0.05 sec	0.02 sec	0.01 sec
samples: 127*120 = 15240	samples: 127*1200 = 152400	samples: 127*2400 = 304800	samples: 127*6000 = 762000	samples: 127*12000 = 1524000
class 1: 107*120 = 12840	class 1: 107*1200 = 128400	class 1: 107*2400 = 256800	class 1: 107*6000 = 642000	class 1: 107*12000 = 1284000
class 0: 20*120 = 2400	class 0: 20*1200 = 24000	class 0: 20*2400 = 48000	class 0: 20*6000 = 120000	class 0: 20*12000 = 240000
class 1 balanced: 12840*0.19 = 2440	class 1 bal: 128400*0.19 = 24396	class 1 bal: 256800*0.19 = 48792	class 1 bal: 642000*0.19 = 121980	class 1 bal: 1284000*0.19 = 243960
Total: 2400+2440 = 4840	Total: 24000+24396 = 48396	Total: 48000+48792 = 96792	Total: 120000+121980 = 241980	Total: 240000+243960 = 483960

Table 7.1.: The input data-set for the different batch sizes. The first row shows the number of data samples for each batches. The second and third row shows the number of data samples separately for the two distinct classes. The fourth row is the positive instances after majority class reduction, and the final row is the total samples after the reduction.

of the feature space, leading to change in data characteristics. We suspect that for the increasing batch sizes the RBF kernel function is somehow influencing the slowdown of the performance degradation. Another reason might be the marginal decrease in the outliers in one or more features is countering the slowdown of the classifier performance. However, a more intensive probe is needed to determine the probable causes, a task reserved for the future.

The sensitivity and specificity values obtained for each classifier is pivotal to finding out the minimum Bayesian window required to obtain our desired accuracy of 99.9%. An intuitive look into the table below (7.2) will tell us that higher accuracy metrics will tend to use less iteration for the Bayesian probability calculation. We can consolidate this idea by looking at Figure 7.1. For calculating the minimum Bayesian window we take the assumption that given the distinct batch sizes and classifiers, it will not take more than 10 predictions to reach our desired Bayesian accuracy. This makes it possible to fix the sample size for our experiment. The minimum window sizes for each batch is consistent with our previous findings regarding the classifier accuracies. With the increase of batch sizes, as the accuracy goes down, it takes a longer prediction window to reach our desired target of 99.9% accuracy. Though its worth noting here that in terms of absolute time the smaller batches will be quicker owing to the data collection time. However, for a particular batch size, all the classifiers perform similarly when treated with the Bayesian model. We can argue that our Bayesian Inference method works pretty well irrespective of the selection of the classifier, and even the increase of batches does not severely affect the minimum window size. This in essence once again reflect the individual classifiers. A weak learner with a low accuracy score will take more windows to achieve the target, however we can always en-

Batch Size	Classifiers	ROC Score	Sensitivity	Specificity
1 sec	RF	0.889	0.867	0.909
	SVM	0.813	0.801	0.825
	XGB	0.886	0.851	0.922
0.1 sec	RF	0.805	0.769	0.840
	SVM	0.770	0.726	0.825
	XGB	0.792	0.753	0.832
0.05 sec	RF	0.769	0.734	0.807
	SVM	0.760	0.714	0.807
	XGB	0.771	0.727	0.815
0.02 sec	RF	0.747	0.715	0.780
	SVM	0.742	0.686	0.799
	XGB	0.738	0.682	0.795
0.01 sec	RF	0.742	0.718	0.766
	SVM	0.757	0.721	0.793
	XGB	0.686	0.644	0.729

Table 7.2.: The performance measures for different batch sizes for all three classifiers. We can notice a gradual reduction in accuracy with the increased number of batches.

sure that we reach that target with the Bayesian model. This whole idea that we can get away with a single weak learner if it means better resource usage is a step up to our goal in achieving incremental resource optimization whenever possible. It can be noted that the Bayesian model itself use some hardware resources, and a trade-off needs to be achieved to find the optimal balance between the resource utilization of both our classifier and the Bayesian model. While the selection of the batch size appears to be straightforward given we have the quantitative measures to base our decision upon, the selection of the optimum classifier is a slightly trickier one. In the next segments we will do some more evaluations to see what works best for our use case.

7.2 Resource consumption for different prediction models with Bayesian Inference

In the previous section we established the batch size that complements both our data-set and prediction models and provides an optimal configuration. On top of that we have established the minimum Bayesian window required for each classifier for the different batch sizes. However, this is only a part of our total optimization scheme. Another important aspect is the resource utilization for the different models along with the BI. Different classifiers, owing to their internal architecture and logic, will have different resource consumption for both training and prediction. For our use case we enforce a pre-trained model to predict on the new data instances, and as a result are only concerned with the resource consumption during prediction. To reiterate, the prediction is done on the edge devices, which is likely to be a low powered device with CPU and power constraints. This ensures that we provide a level of flexibility when deciding on the edge hardware, and also keeping the future in mind if and when an upgrade/downgrade is required.

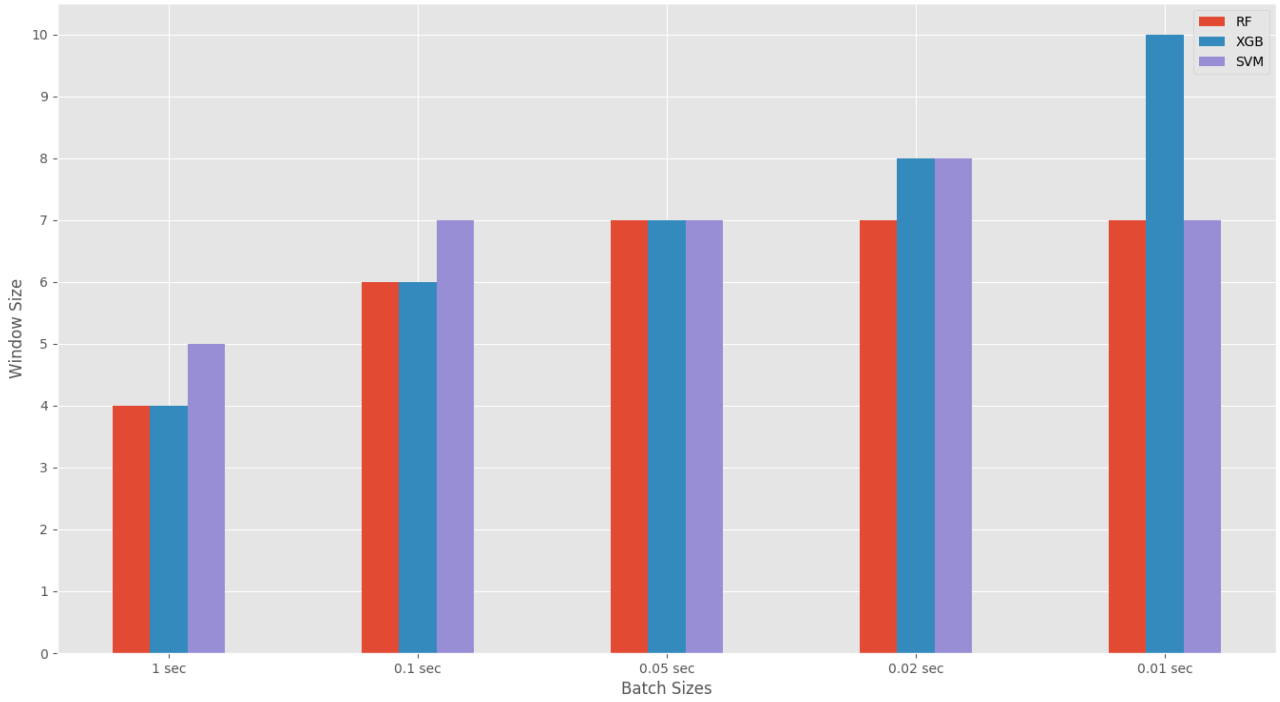


Figure 7.1.: The minimum Bayesian window required for the different batch sizes. We can see the window size is directly proportional to the batch sizes. On the contrary the classification speed for the individual instances will be faster for the smaller batch intervals, but it is not relevant to our use case.

The three classifiers we have used vary greatly from one another in terms of implementation logic, and will have distinct footprints on the hardware. For example RF, which is an ensemble of decision trees, has a very fast training time but is much slower at the time of predictions. Ensemble classifiers in general tend to be heavier on the memory consumption. A collection of several classifiers working in parallel, no matter how optimized, often have higher memory footprint than a single learner. On the other hand, a single classifier based on the principle of linear separation (like SVM) should theoretically have a lower prediction time, as it only needs to check which side of the decision boundary to place a newly arrived data instance. The XGB algorithm can take advantage of fast multi-core processors to significantly reduce the execution times, an enhancement however not possible given our current hardware setup. It is also worth mentioning that the size and characteristics of the input data-set also has a big role to play on these factors, as well as the implementation logic behind these algorithms. Different implementations of these ML algorithms (Scikit-Learn, Weka [19], R [28] etc) will have slightly different programming constructs even for the same implementation logic, and can lead to a different footprint. For our implementation we have used the Scikit-Learn [39] module, based on the Python programming language.

Although we have already decided on the batch size, it still makes sense to invest some time to find out the resource consumption for each batch interval. This can further consolidate our claim that the 1 sec batches are the optimal solution given our sample selection of batch sizes. We only conduct this experiment with the RF classifier, but we expect similar patterns for the

other models. The results show a steady increase in memory with increase in batch sizes. This is somewhat expected, as reducing the batch duration from 1 sec to 0.1 sec would generate 10 times more batches, and hence the gradual increase in memory for feature extraction and prediction. However, it is interesting to note that the BI model maintains a somewhat steady footprint, and only shows a marginal spike in memory usage for the increasing batch sizes.

For the next part, we look into the memory consumption of the three classifiers for a single batch of 1 sec duration that represents our actual use case. We observe that for the given batch size, all three classifiers have a similar footprint on the edge device, and can be used interchangeably with respect to the changing requirements. However, for the execution times ensemble learners appeared slightly faster compared to SVM for the prediction, but all three exhibit somewhat similar results for the Bayesian execution. We can interpret the results to support our primary goal throughout the entire research, i.e. to provide flexibility to the entire system whenever applicable without compromising the overall accuracy of the predictive model.

7.3 Resource consumption of Quantized vs Non-quantized signal

In Chapter 4.4 we introduced the concept of signal quantization, and used it to reduce the bitrate of our data-set from 100 KHz to 500 Hz. We then moved on to show that the re-sampled data obtained from this quantization process closely follows the original signal, and even produces similar classification results. In this section we will observe the effect of the quantization process on the hardware memory. In theory both should have the same memory footprint. This is because we only reduce the bitrate for the individual batches, but the number of batches remain the same for both. For example, in case of the original non-quantized data, each batch is computed over 100,000 data points, while we extract the features for that batch. On the contrary for the quantized data we compute each batch based on the reduced (500) data points, but in both cases we get a single batch of data. This, when computed over the entire data-set, will give the same number of batches for the fixed batch interval of 1 sec. The results obtained matches our theoretical assumptions, and shows similar memory usage for feature extraction, prediction (using RF) and BI for both the quantized and non-quantized signals collected over 1 sec batches. While the signal quantization does not seem to affect the gateway, it can be a crucial component when selecting the DAQ. There is a considerable difference in price of two data acquisition systems with sampling rates of 100 KHz and 500 Hz respectively.

To sum it up, we now have a quantitative measure of the optimization parameters best suited for our scenario. Selecting the batch interval to 1 sec gives the highest accuracy for the classifiers. Any one of the three classifiers can be chosen for the prediction task, and our Bayesian model takes a maximum of 5 batches for the chosen interval to achieve the level of certainty for the predictions. Moreover, the different individual processes have a low enough memory footprint to be implemented on most hardware at different price points and capacity. Finally, we have an added option of resampling the data to reduce the bitrate of the original signal. Next, we can implement these configurations to a test setup that resembles the real world use case for our application.

8 Proof of Concept

In this segment, we form a prototype as a proof of concept for our design principles by briefly describing each of the components in the implementation pipeline.

8.1 Hardware Overview

In the previous chapters we have discussed about creating, testing and improving ML models to solve a very specific problem related to our use case. The next step would be to implement that model by creating an end-to-end solution responsible for data collection and modelling to visualizations of the predicted outcomes. Our main goal, throughout the entire implementation has been to achieve a high degree of flexibility. A good ML model should be able to adapt to changes in hardware and environment. Our model is in no way restricted to the devices and tools we use for the implementation, and can easily be upgraded or replaced in the future.

In Chapter 3.2 we mentioned the data collection mechanism where the hydrophones send the acquired signal to a DAQ or a collection of DAQs. The next logical step would be to have a central gateway to collect these signals. The gateway is perhaps the only piece of hardware we add to the existing setup, and is central to our entire design principle. Figure 8.1 shows the different components of the implementation. We look into each of them in a bit more detail.

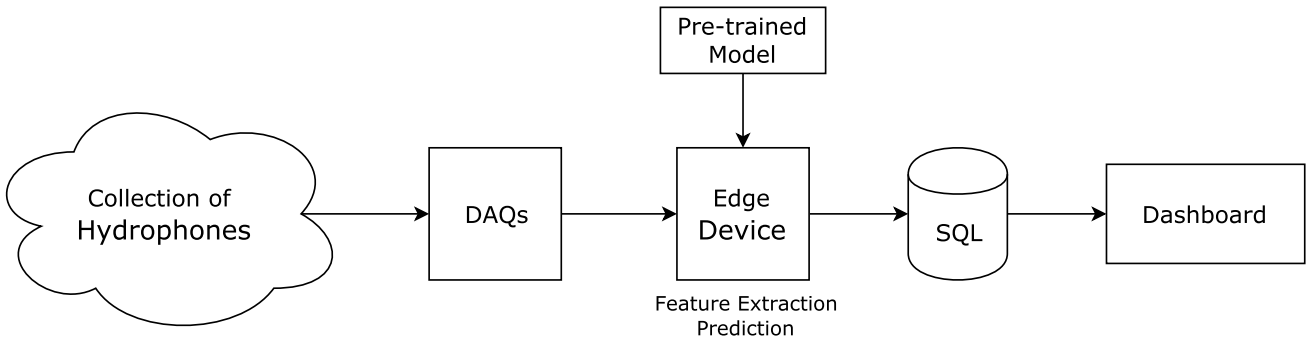


Figure 8.1.: The abstract schematic for the implementation process and the different components.

Each reservoir has a minimum of four sensors for collecting raw acoustic data. The signals are processed through a pre-amplifier for noise reduction before moving into the data acquisition tools. Ideally, each reservoir data is collected by a separate DAQ, and finally sent to the edge device which constitutes the heart of our pipeline.

Gateway/Edge Device

As mentioned, the gateway is a key component where almost all the workload needs to be handled. They are located on site and ideally close to the DAQs to keep the latency as low as possible when dealing with high amounts of raw data. We had two separate design considerations for selecting the hardware for the gateway. One would be to have the gateway as a naive

device with a simple function to collect and aggregate the data from multiple DAQs before sending them to a remote server for processing. The second, and the one implemented by keeping bandwidth consumption in mind, is to have the gateway do all the machine learning tasks and only send the predicted outcome to a remote database. We chose Raspberry Pi 3 model B [44] as the hardware for the gateways. These kits provide a highly configurable solution and at the same time being robust and easily movable (when inside a casing), while having a small physical footprint. This makes it an ideal option for our use case.

8.2 Implementation Pipeline

We now look into the implementation pipeline in details, where we divide the tasks into a group of functions to make the process streamlined and modular. There are four major functions, each executed sequentially after one another. The first function takes care of the data preprocessing (batching, resampling, class reduction) and feature extraction. It collects 1 sec of data, extract the features for that set and would then move on to the next 1 sec batch. This type of learning can be considered to be a hybrid model, where the classifier is trained earlier on a static dataset, but the prediction is carried real time on live incoming data.

The output, which is the test set for our predictive model, is then fed to the second function where the prediction is done using a previously trained classifier loaded into the gateway. This gives us two distinct advantages. Apart from making the predictions (and the entire process in general) faster and more real time, it also requires less computation which gives us the flexibility to scale down in terms of hardware if required. However, this also has its own share of downsides, which we will discuss in the later section. We can also add a second layer of optimization if required. In Chapter 7.1 we found out that the minimum Bayesian window for all the classifiers to achieve 99.9% accuracy does not exceed 10 for any of the different batch intervals, and is in fact half of that if we only consider the 1 sec batches. This means that collecting just 5 batches from the input stream will be sufficient to get the desired results, thus conserving time, resource and bandwidth.

The third function is responsible for applying the Bayesian models to the predictions and generates a flat file with all the relevant data. However, instead of the separate batches, it provides a macro view in terms of the different reservoirs. This file also includes the error checking mechanisms and the probabilities associated with each Bayesian inference.

Finally, the fourth function creates a database connection and uploads the obtained data to an online relational database. The choice of the database depends on some basic requirements: it should be fast, reliable, secure and scalable. Most modern relational databases can be said to have these things in common, though some may offer added functionality over other. We decided to use Microsoft SQL server [50], though any other product would have sufficed just as well. The results can be visualized with the help of either custom built dashboards or available Business Intelligence tools that provide ready-made visualizations for a large variety of data type and sources. We opted for the ready-made solution (Microsoft Power BI) for the time being to make the roll-out faster.

8.3 Implementation Challenges

In the previous section we briefly mentioned about the downside of using a pre-trained classifier on live data. A model trained with a static data is ideal only if the new data has the same characteristics. However, intrinsic or extrinsic changes in the setup may lead to variations in the incoming data, causing a decline in prediction performance. We can say this issue is trivial to our use case, since we use Bayesian methods to reduce prediction errors.

Looking into Figure 8.1, we can see that the gateway is the single point of failure. The gateway acts as the lone bridge between the on-site and the control center, which can be hundreds of miles apart. To solve this, we can introduce additional gateways for redundancy and load balancing. However, we will not go into different architectures or procedures to achieve redundancy in this thesis, and prefer to leave it for future work.

We can also argue that we anticipated some of these challenges at the very beginning, and tried to create fixes along our way throughout the implementation process. The BI model, for example ensures a high level of accuracy and tolerance to data variations even with an offline training model. In a similar way, we induce flexibility while selecting the DAQs and the gateway. The option to quantize the original signal from 100 KHz to 500 Hz means that the existing DAQs can be swapped with cost effective alternatives. Yet, it is impossible to anticipate and prepare for all the post deployment challenges, something that can only be managed via incremental improvements.

8.4 Data Visualization

In the end, we need a simple yet informative platform to visually represent the results. As mentioned earlier, we use a third party dashboard as a conceptual design idea, which will be replaced by a custom built visualization tool in the future. Our choice of the third party dashboard is based on some fundamental requirements like ease of implementation, security, compatibility with various data sources and long term support.

Figure 8.2 shows the overall design of the dashboard and its primary components. We can see it consists of five distinct but related sections. The top part shows the prediction outcomes for each new specimen, with the leaked occurrences represented as vertical bars. The bottom left part of the dashboard contains a list of the incoming data and their corresponding leak probabilities. We can individually select each specimen to check the status of that particular one. The predicted outcome is represented in terms of both the Bayesian output and the ratio (explained in Chapter 6.3). This latter adds a level of certainty to the predictions, and can ease the decision making process. The selection panel (bottom middle) lets the user select only a particular class of data from the entire data-set, that can help the user to quickly distinguish the leaked instances from the data-set. We tried to avoid excess complexity while deciding on the components of the dashboard, and all of them can be considered as self explanatory.

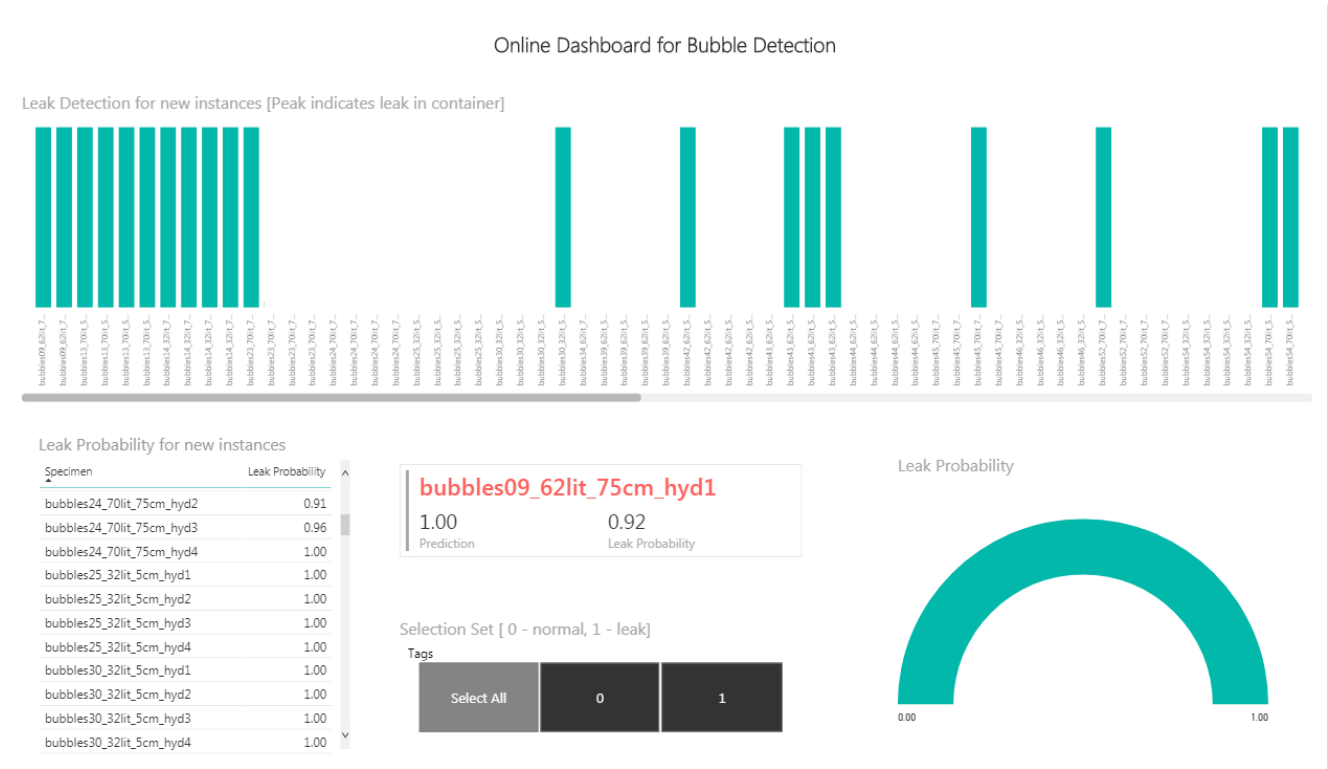


Figure 8.2.: Platform for visualizing the predicted outcomes.

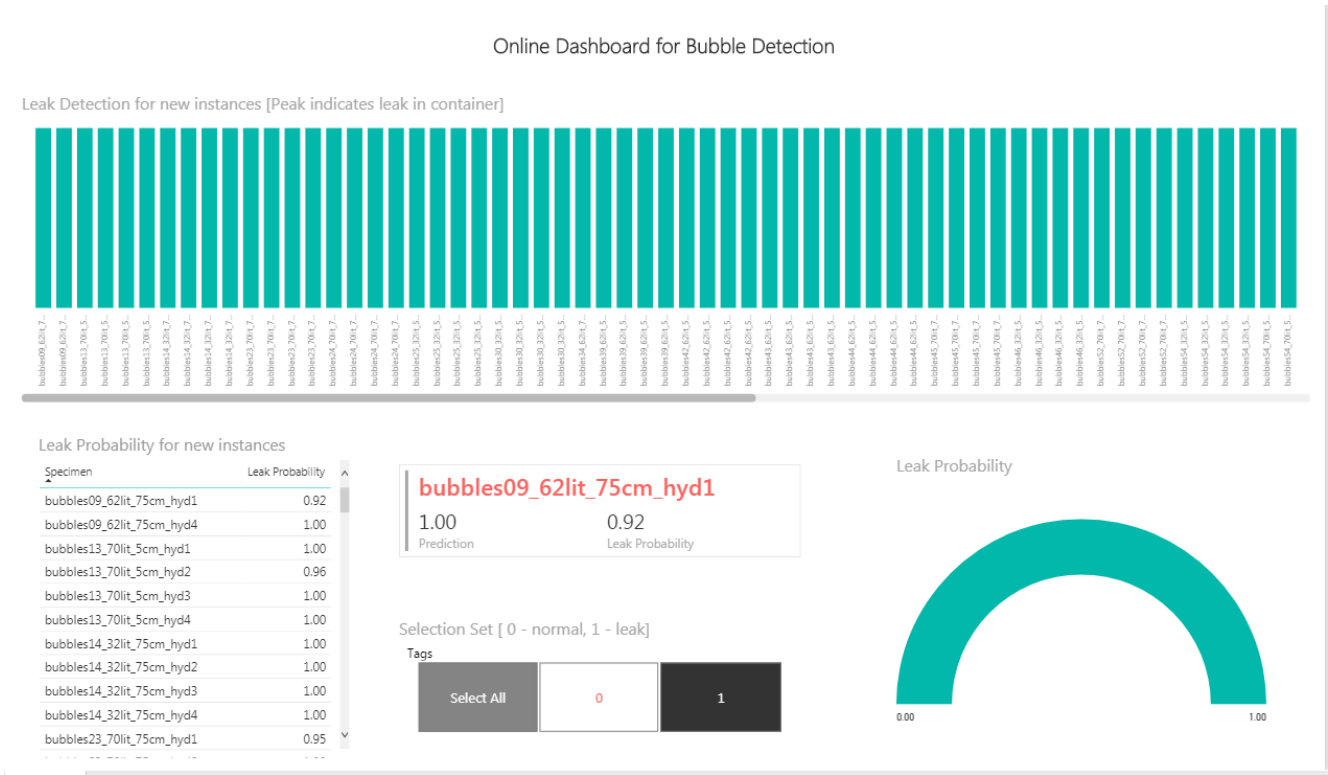


Figure 8.3.: The selection set panel gives the option to display a particular class of data.

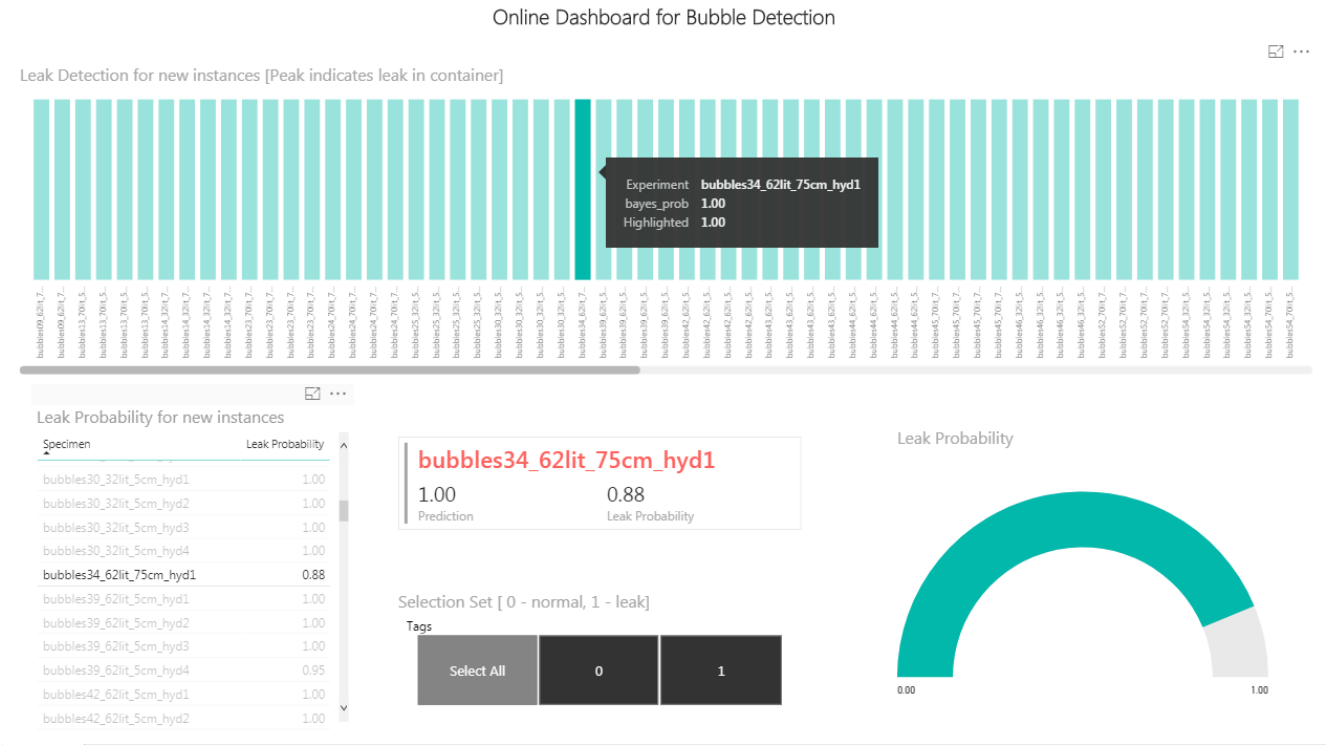


Figure 8.4.: A single specimen can be selected from the list for further investigation.

9 Related and Future Work

We conclude by presenting some related work to our use case, followed by the impending work to be undertaken for our implementation.

9.1 Pertinent research in condition monitoring and predictive maintenance

We will look into some other works over the years that are related to our use case. It does not need to be strictly restricted to the monitoring and maintenance of industrial fluid systems, but our research for related work can be extended to interesting methodologies of condition monitoring in general. With IoT gradually becoming a more matured and stable platform over the years, different methodologies are tried and introduced to make the process of predictive maintenance fully automated, self-configurable and self-optimizing. Most of them are based on ML or deep learning models, or at least some form of cognitive learning. However, it will not be complete to end the chapter without mentioning the areas of future work for our implementation. Improvement in terms of flexibility, optimization and overall performance is an ongoing process, and we will try to list some of them here before finally concluding the thesis.

Rostek et al.[63] in their research documents the use of Artificial Neural Networks (ANN) to predict leaks in fluidized-bed boilers in an industrial power plant. The pipes are exposed to the temperature and other environmental changes, which result in the sedimentation of water particles on the inner surfaces of the pipes. This results in the gradual overheating of the pipes, and an early detection is necessary to keep the repair cost lower. The data-set consists of control information of a single boiler collected over a period of seven years. These normally include the steam temperature and pressure, flux, efficiency etc. The faults are divided into three distinct categories depending on the location of their occurrence in the setup. They also work as labels for the training data, making it a supervised learning problem. The authors inferred that ANN (with the help of neurons) can be used for modelling problems like this which are otherwise difficult to represent. Statistical tools are used to autonomously create the ANN models with two distinct configurations, multi-layer perceptrons (MLP) and RBF. A custom design environment (called statistical ANN or SANN) is utilized to pick from a range of training algorithms. The authors applied two different methods for fault diagnosis. The first uses a residual method of detection. The idea is to first design a model for the fault-free state, and then apply that to find out the residual values. The residual value r is defined to be the difference between the measured signal y_s and the outcome of that (fault-free) model $\hat{y}_s$. For a fault-free state these two values should be identical, resulting in r being 0. Similarly a non-zero value of the residual function indicates leak in the boilers. The model is trained with data representing the working state of the system (no leaks) aggregated over different time periods and oversampled. The second method indicates isolation of leaks. It uses pattern recognition mechanisms to check whether a system deviates from its normal behaviour, and decides if a leak exists or not. The ANN is selected as a function of the residual values and the subset of variables sensitive to leaks. The training set consist of two kinds of data: one representing normal fault-free operation and the other depicting the immediate transition of the boiler from operating to bad state leading

to eventual shutdown. Evaluation of these models were conducted on the variables that are more susceptible to leaks. The results show that faults are predicted 2 to 9 days before a boiler shutdown, which are in line with the author's expected prediction results.

We look at a different application of the residual method. In their paper, Ferrandez-Gamot et al.[59] propose a data-driven approach to identify leaks and contamination in drinking water distribution networks (WDNs). The monitoring of the WDNs are based on IoT, with localized sensors collecting pressure measurements that make up the data-set. A similar residual function like the previous use case is obtained by comparing real world data to a simulated model representing a leak-free setup. A hydraulic simulator is applied to generate values for the leak-free scenario in terms of known parameters and historical data. The methodology combines the use of models to obtain the residuals and then use classification algorithms to predict on those residuals. At the initial step, the various residuals are obtained for the different leak scenarios which constitute the training set for the current problem. Each feature can be labelled as leak or non-leak based on the residual value obtained for that particular pair, making it a supervised learning problem consisting of multiple class labels. Care has been taken to make the residual feature set generalize to the real world scenario, capturing different characteristics of the two kinds of data (simulated and real). The authors defined a precision index bigger or equal to a certain threshold as an acceptable accuracy metric for their model. The methodology is implemented on a test WDN termed as the Hanoi water network, made up of a single reservoir and 31 junction nodes. A finite impulse response (FIR) sensor is applied to differentiate between the various leak scenarios in order to improve the leak localization results. The authors observed two distinct parameters: the scaling ratio and the order of the filter to determine the maximum values after which there is no visible improvement of the prediction results. While an optimal value for the filter order was obtained, for the scaling ratio a set of values is proposed instead of a fixed value to better represent the different leak ranges in the data-set. The benchmarking results show a location accuracy of 83% for measurements with 2% noise and no uncertainties. The overall accuracy with noise and uncertainties on factors like leak and demands are also found to be higher than the other methodology (Angle method) which is used as a basis of comparison for the classification model.

We can also find alternative methods for the previous problem of leak detection in WDNs. Rajeswaran et al.[61] propose the application of a graph partitioning algorithm to optimize the usage of flow measurements, saving time and measurement cost associated with it. They argue that most of the existing leak detection methods are either expensive, difficult to implement, time consuming or affects the operation runtime. These factors can be minimized to a large extent if we can employ methods to narrow down the location of the leak within the WDN. The measurements are only based on the flow of water, which can be obtained by an on-demand investigation of the pipes. However, checking each pipe individually has cost associations, and it is desired to keep the checks to a minimum. The authors propose a simple yet effective method to achieve this. For the WDN visualized as a network of nodes, and assuming some node in the network has leaks, the following algorithm can be applied.

1. Select some edges that roughly divides the network into two separate subnetworks.
2. Check the water balance to trace the leak in one of the subnetworks.
3. Divide that subnetwork into two separate networks and check the water balance again.

4. Repeat the above steps until a small portion of the network is identified to have the leak.

The above methods help to keep the on-field checks limited and to the point. For the algorithm, the authors decided to use a divide-and-conquer approach, where graph partitioning is used in succession with flow measurements to achieve the desired solution of leak detection. The flow rate can be measured by aggregating the supply (source) and demand (utilization) nodes in the network. For an ideal leak-free setup the inflow rate should be identical to the outflow rate. On the other hand, an outflow rate lower than the rate of inflow indicates leak in one of the nodes in the WDN. Knowledge about the rate and direction of the flows in the pipes is crucial for proper implementation of the algorithm. The authors propose some alternate methods like hydraulic simulations in case proper flow measurement instruments are not available. Another area of optimization challenge is minimizing the number of iterations, while ensuring that a balanced partition is obtained at each iteration. For the evaluation, several scenarios were considered that include both leaks in the nodes and the pipes for different WDNs at separate geographical locations. The results showed a lower number of queries on the nodes or pipes than a predefined goal set for each location. Next we discuss some of the advantages of this methodology as suggested by the authors. Firstly, the general nature of the implementation process allows it to extend to similar problems beyond leak detection like water theft, etc. In fact the same method can be applied to any kind of fluid transfer systems. The model takes advantage of the existing low cost sensors with minimal addition of new hardware. Secondly, if the network topology of the WDN is known beforehand, the partitioning can be done offline and stored as a decision tree. However, the implementation also takes certain assumptions that make it difficult to generalize. It expects a prior knowledge of the WDN topology, while at the same time assuming the network is steady and not susceptible to churns. The authors also provide the solution for a specialized case which considers only one leak in the network. Multiple leaks at different locations will make the partitioning algorithm more complex. However, a more generalized method for multiple leaks is proposed at the end of the paper.

For the next case study we look into a health monitoring application [64] for batteries that makes use of a Bayesian framework. We find this an interesting case study for its application of a special type of classification algorithm. The need for applying probabilistic models based on past data arises from the difficulty of measuring certain systems (like batteries) with enclosed internal components without disrupting the operations of the entire system or part of that sub-system. The paper deals with providing a prognostic approach to determine the life-cycle of the components based on historical data and measurements of the current state. The learning model used is called a Relevance Vector Machine (RVM) [65]. RVMs can be defined as a Bayesian framework that uses linear models to obtain sparse solutions to classification and regression tasks. The linear models possess the characteristics similar to SVMs, and hence the name. The primary advantage of this model over traditional SVM is that it provides a probabilistic nature to the predictions. Unlike SVM, the target variables are representations of the model itself. Also, SVM handles overfitting by adding some restrictions to the parameters in the form of margins, while in RVM this is achieved by defining a prior probability distribution over the parameters. The probability density function (PDF) for Bayesian frameworks is expressed as a Kalman filter for linear models, which is essentially an estimator based on a collection of measurements over time to produce a more accurate result. However, the authors propose to express the PDF in terms of particle filters (PF) that uses Monte Carlo simulations [58]. The

implemented model is a combination of RVM and PF, each of which is applied to the offline and online analysis respectively. For the offline analysis the features are generated from the frequency response of the battery units. This data is trained using RVM regression techniques to learn the temporal dependencies of the features. This is next applied to the online analysis and detection of faults using PF, which starts the prognosis process to predict the life-cycle of the power cells. The results show the RVM model is non-reactive to outliers in the data, while having an overall improvement in precision over an arbitrarily chosen threshold for fault identification.

We will quickly go through some other implementations of leak detection mechanisms. Verde et al.[66] discuss about a special scenario where the flow measurement tools are attached only at the terminals (inlet and outlet) of the pipeline network. The authors propose a parametric model to solve the problem. They begin with a generic model for leak detection, and then proceed with the identification and development of a parametrized family of models suited to their use case. The argument is that for a 'two leak' position in a pipeline, the model representing the length of the pipeline should be divided at least into three sections in order to isolate the leaks. The next task involves identifying the unknown parameters obtained from the transient parametric model in the previous step, for which a batch identification process is employed followed by the leak detection algorithm. The theoretical metric of improvement is based on minimizing a quadratic loss function. Evaluation on a simulated test-bed yielded estimated error margins to be less than 0.7%, with the quadratic loss function achieving very low values.

The next paper [62] proposes a real-time online learning method based on wireless sensor networks for fault detection and size estimation. The authors introduce a machine learning framework that takes advantage of pattern recognition algorithms. The leak identification is done by measuring the pressure sensors to detect the transient waves that are generated for a short time when a leak appears, a technique known as negative pressure wave (NPW). The problem is treated as a supervised one, with the WSN nodes providing the required labels. The feature set consists of both time and spectral domain characteristics obtained after two separate tests to determine the optimal attributes for the data. Four independent classifiers are used for learning to provide a comparative analysis of their learning capabilities. The mean accuracy for the binary classification is found to be at 94%, with the Gaussian Naive Bayes (GNB) classifier giving the best accuracy.

Liang et al.[60] presents a leak detection mechanism based on analysis of the nodes and the pipeline as a coupled state, instead of treating them individually. Detection mechanisms based on the principle of NPW can result in high rate of false alarms, while having slow computing speed and high resource usage. The authors propose an improvement over the existing NPW methods that utilizes logical reasoning, improving real-time detection of oil spills while keeping computation and resource usage low. The method, called Negative Pressure Wave Logical Reasoning (NPVLR) consists of noise reduction, feature extraction, and recording of irregular pressure changes for different operating conditions like switch off, leaks etc. A threshold analysis is conducted on these values to logically deduce the probability of leaks or theft in the pipeline. The evaluation is done on an industrial setup consisting of five stations over a wide geographical region. Results show accurate detection of leaks (false alarm $<5\%$), with a small response time and location error.

Besides the ones discussed above, there are many interesting ongoing research work in predictive maintenance for an industrial setup. We have only produced a handful of case studies for keeping it concise.

9.2 Future work for our use case

To wrap this up, we discuss some of the future work applicable to our use case. We would like to achieve further optimization for the on-site hardware. For example, our initial plan was to use STM32 ARM microprocessor based boards as the preferred choice for the gateways. However this could not be achieved due to the minimum memory requirements and processing speed for our application exceeding that of the chipset. We would also like to explore the idea of using deep learning algorithms in the future. Artificial Neural Networks have many application areas in condition monitoring, and have proven to be an effective mechanism for detection of faults in industrial systems. This is not applicable to our current setup given the limited size of our sample and the difficulty of collecting large amounts of labeled data, but remains a viable option in the future. As mentioned earlier, the current training data along with the target values are obtained via an experimental setup, with each specimen data collected over a period of 120 seconds. Generating large sets of labeled data is an expensive process. Alternatively, we can use the data generated from the real industrial setup and treat our learning problem as an unsupervised one. We would like to employ some anomaly detection algorithms as a part of our future case study. But a more immediate task can be to improve the feature engineering for our data-set. We have only considered the time domain statistics during feature extraction, and would like to explore the idea of combining both the temporal and spectral features of our input signal to observe any meaningful improvement in classifier performance. Lastly, we will treat optimization as a repetitive process, by constantly monitoring and gathering information about the current performance and find ways to improve on it.

10 Conclusion

We conclude the thesis by summarizing the motivation for our research, the methodologies used and the results obtained. From the very beginning we are tasked with building a custom condition monitoring solution for a very specific industrial use case concerning leak detection in a fluid transfer system. The implementation strategy is governed by limited sensing and data collection capabilities, along with environmental and distance constraints. This makes performance optimization as one of the key deliverables for the implementation process beside the apparent measure of prediction accuracy. We implemented a two-step methodology to provide highly accurate and robust leak detection by introducing a single hardware (gateway) in the existing setup. The first step consists of training a ML model on the labeled data and use that to predict on new data instances. In the second step, we further improve the prediction results by applying Bayesian Inference to determine the probability of leaks for the reservoirs. The optimization is achieved partly by providing the flexibility to reduce the bitrate of the acquired signal to adjust according to the data acquisition tools, and the rest by using a relatively low powered (in terms of CPU and memory) hardware for the gateway. We implemented a prototype of the model on the existing industrial setup, with an option to easily store and visualize the predicted outcomes. The prototype will be used to evaluate the methodologies we have employed in our applied research, and based on the feedback a full-fledged implementation will be conducted in the near future.

Bibliography

[1] Study and application of acoustic emission testing in fault diagnosis of low-speed heavy-duty gears. *Sensors*, 11(1):599–611, 2011.

[2] P A. G. Van Der Geest. The binomial distribution with dependent bernoulli trials. 75:141–154, 02 2005.

[3] Jim Albert. *LearnBayes: Functions for Learning Bayesian Inference*. 2011. R package version 2.12.

[4] Ricardo Barandela, José Sánchez, Vicente García, and E Rangel. Strategies for learning in class imbalance problems. 36:849–851, 03 2003.

[5] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[6] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29, June 2004.

[7] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[8] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *J. Mach. Learn. Res.*, 11:1471–1490, August 2010.

[9] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[10] 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*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[11] Allen Downey. *Think Bayes*. O'Reilly, Sebastopol, CA, 2013.

[12] Alberto Ferrari and Marco Russo. *Introducing Microsoft Power BI*. 2016.

[13] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[14] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

[15] Jerome H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38(4):367–378, February 2002.

[16] Arnulf B.A. Graf and Silvio Borer. *Normalization in Support Vector Machines*, pages 277–282. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[17] Sang Guk Lee, Jong Hyuck Park, Keun Bae Yoo, Sun Ki Lee, and Sung Yull Hong. Evaluation of internal leak in valve using acoustic emission method. 326-328:661–664, 01 2006.

[18] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[19] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[20] Andrew Hamilton-Wright, Linda Mclean, Daniel Stashuk, and Kristina Calder. Bayesian aggregation versus majority vote in the characterization of non-specific arm pain based on quantitative needle electromyography. 7:8, 02 2010.

[21] Trent Hauck. *Scikit-Learn Cookbook*. Packt Publishing - ebooks Account, 2014.

[22] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), IJCNN 2008*, pages 1322–1328, 2008.

[23] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Trans. on Knowl. and Data Eng.*, 21(9):1263–1284, September 2009.

[24] Helbig (Deutsche Post Ag). Henning, Kagermann(National Academy of Science and Engineering). Wolfgang, Wahlster (German Research Center for Artificial Intelligence). Johannes. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *Final report of the Industrie 4.0 WG*, (April):82, 2013.

[25] Taeho Jo and Nathalie Japkowicz. Class imbalances versus small disjuncts. *SIGKDD Explor. Newsl.*, 6(1):40–49, June 2004.

[26] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001.

[27] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.

[28] Brett Lantz. *Machine Learning with R*. Packt Publishing, 2013.

[29] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. 23, 11 2001.

[30] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Mach. Learn.*, 40(3):203–228, September 2000.

[31] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *Trans. Sys. Man Cyber. Part B*, 39(2):539–550, April 2009.

[32] A complete tutorial on tree based modeling from scratch (in r & python). `https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/`. (Accessed on 12/21/2017).

[33] Complete guide to parameter tuning in xgboost (with codes in python). https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/. (Accessed on 12/22/2017).

[34] Classification and regression trees for machine learning - machine learning mastery. https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/. (Accessed on 12/21/2017).

[35] Discover feature engineering, how to engineer features and how to get good at it - machine learning mastery. https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/. (Accessed on 12/22/2017).

[36] Data Mining. Springer Series in Statistics The Elements of. *The Mathematical Intelligencer*, 27(2):83–85, 2009.

[37] Joseph O. Ogutu, Hans-Peter Piepho, and Torben Schulz-Streeck. A comparison of random forests, boosting and support vector machines for genomic selection. *BMC Proceedings*, 5(3):S11, May 2011.

[38] Tuning the hyper-parameters of an estimator. http://scikit-learn.org/stable/modules/grid_search.html. (Accessed on 01/22/2018).

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011.

[40] Rubiane M. Pires and Carlos A. R. Diniz. Correlated binomial regression models. *Comput. Stat. Data Anal.*, 56(8):2513–2525, August 2012.

[41] D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[42] William Press, Saul Teukolsky, William Vetterling, Brian Flannery, Eric Ziegel, William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes: The Art of Scientific Computing*, volume 29. 1987.

[43] A.K. Rao. Acoustic Emission and Signal Analysis. *Defence Science Journal*, 40(1):55–70, 2013.

[44] Raspberry pi 3 model b. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. (Accessed on 12/27/2017).

[45] Claude Sammut and Geoffrey I. Webb, editors. *Samuel's Checkers Player*, pages 881–881. Springer US, Boston, MA, 2010.

[46] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[47] Robert E. Schapire. *Explaining AdaBoost*, pages 37–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[48] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[49] Sklearn dummy classifier — scikit-learn 0.19.1 documentation. `http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html`. (Accessed on 12/23/2017).

[50] Sql server documentation | microsoft docs. `https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation`. (Accessed on 12/27/2017).

[51] A Terchi and Y H J Au. Acoustic emission signal processing. 34, 10 2001.

[52] Ivan Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976.

[53] Sampling: What nyquist didn't say, and what to do about it. `http://www.wescottdesign.com/articles/Sampling/sampling.pdf`. (Accessed on 12/23/2017).

[54] Leland Wilkinson. Revising the pareto chart. *The American Statistician*, 60(4):332–334, 2006.

[55] I. H. Witten. *Data mining : practical machine learning tools and techniques*. Morgan Kaufmann, Burlington, MA, 2011.

[56] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.

[57] M. Zhu. When is the majority-vote classifier beneficial? *ArXiv e-prints*, July 2013.

Related Work References

[58] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Trans. Sig. Proc.*, 50(2):174–188, February 2002.

[59] Lise Ferrandez-Gamot, Pierre Busson, Joaquim Blesa, Sebastian Tornil-Sin, Vicenç Puig, Eric Duviella, and Adrià Soldevila. Leak localization in water distribution networks using pressure residuals and classifiers. *IFAC-PapersOnLine*, 48(21):220 – 225, 2015. 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015.

[60] Wei Liang, Jian Kang, and Laibin Zhang. Leak detection for long transportation pipeline using a state coupling analysis of pump units. *Journal of Loss Prevention in the Process Industries*, 26(4):586 – 593, 2013.

[61] Aravind Rajeswaran, Sridharakumar Narasimhan, and Shankar Narasimhan. A graph partitioning algorithm for leak detection in water distribution networks. *Computers Chemical Engineering*, 108(Supplement C):11 – 23, 2018.

[62] Sidra Rashid, Usman Akram, and Shoab A. Khan. Wml: Wireless sensor network based machine learning for leakage detection and size estimation. *Procedia Computer Science*, 63(Supplement C):171 – 176, 2015. The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)/ The 5th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2015)/ Affiliated Workshops.

[63] Kornel Rostek, Łukasz Morytko, and Anna Jankowska. Early detection and prediction of leaks in fluidized-bed boilers using artificial neural networks. *Energy*, 89(Supplement C):914 – 923, 2015.

[64] B. Saha, K. Goebel, S. Poll, and J. Christophersen. Prognostics methods for battery health monitoring using a bayesian framework. *IEEE Transactions on Instrumentation and Measurement*, 58(2):291–296, Feb 2009.

[65] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.*, 1:211–244, September 2001.

[66] C. Verde, L. Molina, and L. Torres. Parameterized transient model of a pipeline for multiple leaks location. *Journal of Loss Prevention in the Process Industries*, 29(Supplement C):177 – 185, 2014.

List of Figures

List of Tables

A Complementary Figures and Tables

Table A.1.: Data-set metadata

Total size	7.66 GB
Specimens	127
Leaked specimens	107
Leak free specimens	20
Tank capacities (litres)	32, 62, 70
No. of hydrophones	4
Hydrophone distances from base (cm)	5, 75
Data collection duration (each specimen)	120 sec
Sampling rate (data collection)	100 KHz
Total data points (each specimen)	12,000,000

Figure A.1.: The entire data-set for our experiment constituting of 127 binary files

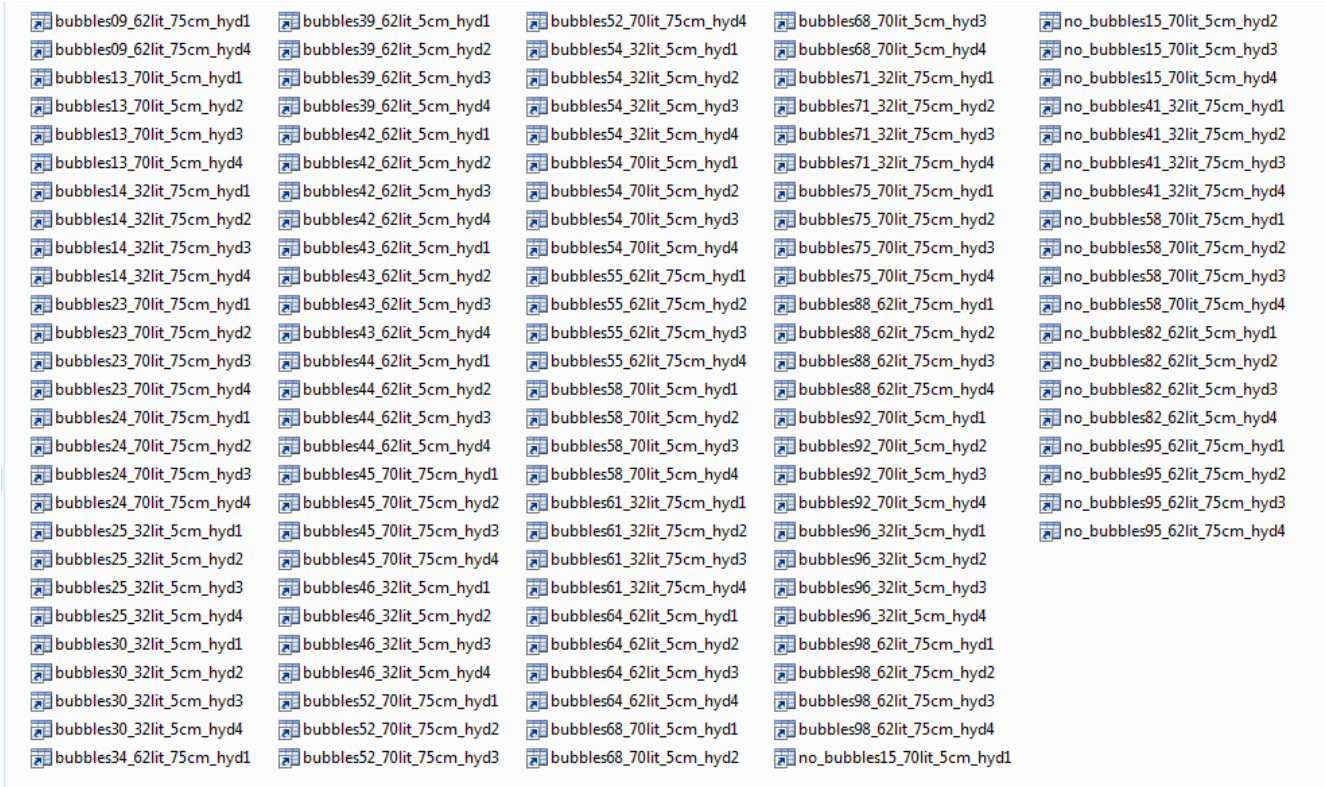


Table A.2.: Sample excerpt from the training set

capacity	mic_distance	mean	std_dev	skewness	kurtosis	rms	peak-to-peak	crest_factor	label
32	5	0.00076515083409	0.978354635722	-0.002971829919391025	-1.5413364082613066	0.978354934926	3.191806212	1.59230096705	1
62	5	0.00119801642025	0.755634185858	-0.001727105934126024	-1.5428763919924862	0.906556168816	3.142352997	1.73002436247	1
70	75	0.00056110350289	0.765533302523	0.0036581846622421140	-1.5671083827499686	0.765533508155	2.419183396	1.56350832883	1
32	75	-0.00098411322735	0.743689201079	0.0010526223618512480	-1.5837730566411512	0.743689852210	2.327045283	1.5798516539	1
32	75	0.00033325774567	0.657932231871	0.0001160201369856775	-1.5553634153114122	0.657932316273	2.150739482	1.61278287714	1
70	5	0.00066410993952	0.976602157030	-0.001609559922199995	-1.5471210667765183	0.976602382835	3.161177124	1.61966068464	1
62	75	-0.00010051510002	0.764083306414	-0.056671112892986260	-1.5624829499211002	0.764083313025	2.452977998	1.59234555874	1
62	5	-0.00113849640852	0.001381083031	0.007242141550866568	-0.6139544998129653	0.001789850388	2.866802464	1.75038429685	1
70	5	-0.00392715844638	0.937057962609	0.001807965256354098	-1.5412895327384717	0.937066191825	3.220520982	1.7329399931	1
62	75	-0.00118468702446	0.00117034951579	-0.033861026230760864	-0.29647051141312450	0.00166529316789	0.009482130	2.14057084286	0
62	75	-0.00118581223722	0.00116038143583	-0.045055717174744990	-0.38035084401694297	0.00165910690993	0.010114272	2.52956574100	0
62	75	-0.00118591021923	0.00118643143912	-0.049601632528779750	-0.32660410805457340	0.0016774989144	0.009798201	2.12499571201	0
70	75	-0.00118842614439	0.00118833949724	-0.056720531322329244	-0.31190012781264587	0.00168062710361	0.010430343	2.12104040947	0
70	75	-0.00118604929047	0.00119359707076	-0.048964766856581576	-0.33652517130324580	0.00168267254294	0.009482130	2.30630078103	0
62	5	-0.00117889660374	0.00116226647174	-0.041968031886059630	-0.38502270226784185	0.00165549405122	0.010746414	2.15324120154	0
70	75	-0.00117776823027	0.00116981933188	-0.041300812477289200	-0.35877981966442807	0.00166000460044	0.010114272	2.52819781276	0
62	5	-0.00118432670352	0.00116933921161	-0.048049563432522020	-0.36028158625062100	0.00166432687069	0.010430343	2.33172285346	0
62	5	-0.0011907050163	0.00116563487698	-0.054049927432934040	-0.36429671713599543	0.00166627821875	0.009798201	2.13930540524	0

Figure A.2.: Flowchart for the leak detection methodology

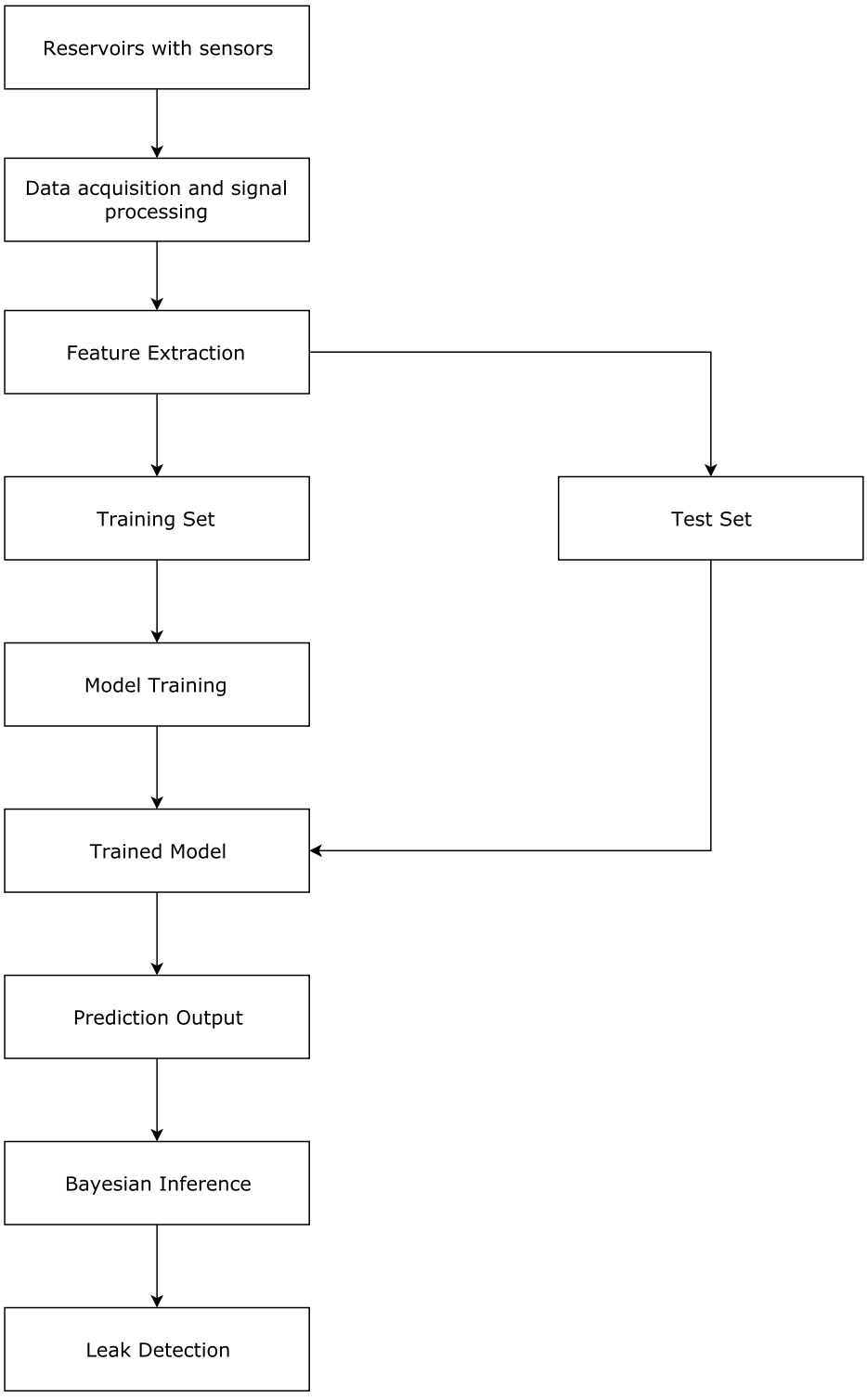


Table A.3.: The Bayesian inferred accuracy for the different classifiers upon a predefined prediction window of size 10. The shaded values indicate the corresponding window to reach the accuracy of 99.9%. Each table represents a different batch interval.

	Accuracy Score		
Window Size	SVM	XGB	RF
1	0.805	0.860	0.872
2	0.945	0.974	0.979
3	0.986	0.995	0.996
4	0.996	0.999	0.999
5	0.999	0.999	0.999
6	0.999	0.999	0.999
7	0.999	0.999	0.999
8	0.999	0.999	0.999
9	0.999	0.999	0.999
10	0.999	0.999	0.999

Table A.4.: 1 sec

	Accuracy Score		
Window Size	SVM	XGB	RF
1	0.803	0.771	0.784
2	0.894	0.919	0.929
3	0.960	0.974	0.979
4	0.986	0.992	0.994
5	0.995	0.997	0.998
6	0.998	0.999	0.999
7	0.999	0.999	0.999
8	0.999	0.999	0.999
9	0.999	0.999	0.999
10	0.999	0.999	0.999

Table A.5.: 0.1 sec

	Accuracy Score		
Window Size	SVM	XGB	RF
1	0.738	0.749	0.752
2	0.888	0.899	0.902
3	0.957	0.963	0.965
4	0.984	0.987	0.988
5	0.994	0.995	0.996
6	0.998	0.998	0.998
7	0.999	0.999	0.999
8	0.999	0.999	0.999
9	0.999	0.999	0.999
10	0.999	0.999	0.999

Table A.6.: 0.5 sec

	Accuracy Score		
Window Size	SVM	XGB	RF
1	0.717	0.714	0.732
2	0.866	0.862	0.882
3	0.942	0.939	0.953
4	0.976	0.975	0.982
5	0.99	0.989	0.993
6	0.996	0.995	0.997
7	0.998	0.998	0.999
8	0.999	0.999	0.999
9	0.999	0.999	0.999
10	0.999	0.999	0.999

Table A.7.: 0.02 sec

	Accuracy Score		
Window Size	SVM	XGB	RF
1	0.739	0.671	0.730
2	0.889	0.807	0.880
3	0.958	0.895	0.952
4	0.984	0.946	0.981
5	0.994	0.972	0.993
6	0.998	0.986	0.997
7	0.999	0.993	0.999
8	0.999	0.996	0.999
9	0.999	0.998	0.999
10	0.999	0.999	0.999

Table A.8.: 0.01 sec

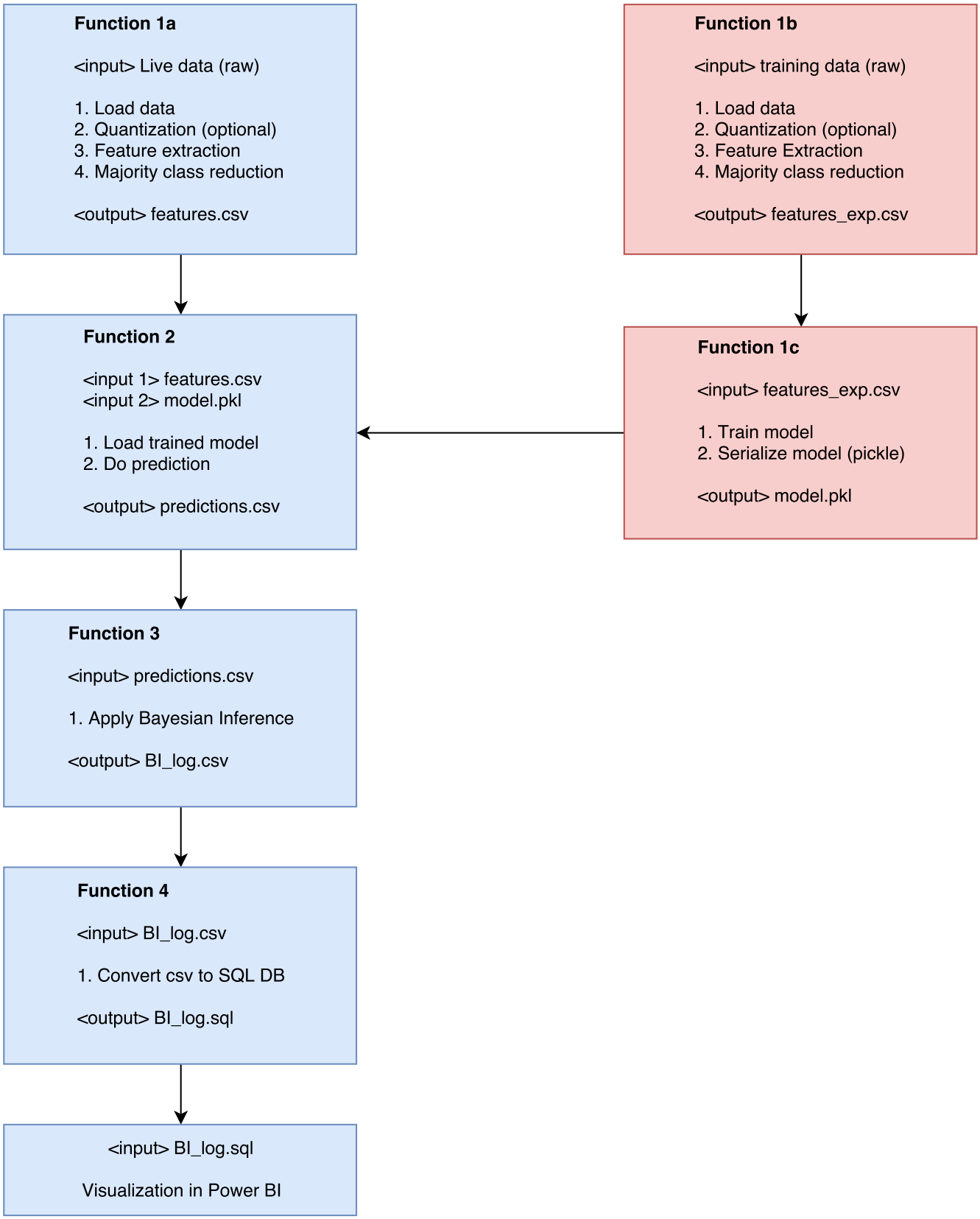
Table A.9.: Sample excerpt from the prediction output log with the error checking mechanism

Specimen	Label	BI	Error	Ratio	Maj. vote
bubbles39_62lit_5cm_hyd1	1	1	0	0.99	1
bubbles39_62lit_5cm_hyd2	1	1	0	1	1
bubbles39_62lit_5cm_hyd3	1	1	0	0.85	1
bubbles39_62lit_5cm_hyd4	1	1	0	0.96	1
bubbles42_62lit_5cm_hyd1	1	1	0	1	1
no_bubbles82_62lit_5cm_hyd4	0	0	0	0.041	0
no_bubbles95_62lit_75cm_hyd1	0	0	0	0.058	0
no_bubbles95_62lit_75cm_hyd2	0	0	0	0.008	0
no_bubbles95_62lit_75cm_hyd3	0	0	0	0.05	0
no_bubbles95_62lit_75cm_hyd4	0	0	0	0	0

Table A.10.: Gateway hardware specifications

Make	Raspberry Pi 3 model B
SoC	Broadcom BCM2837
CPU	Quad-Core ARM Cortex-A53, 1.2 GHz
GPU	400 MHz Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	micro-SD
GPIO	40-pin header, populated
Ports	HDMI, 3.5mm analogue audio-video jack, 4 x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

Figure A.3.: The final implementation pipeline represented as separate modular functions. The blue shaded portions represent the online part of the process, while the red blocks represent the offline learning.



B Code Snippets

A simple implementation of the Bayes' Theorem

```
global predictions

def Bayesian_Inference(predictions, prior=0.5,
                       sensitivity=0.867, specificity=0.909):
    """Function for Bayes' theorem.
    input params: prediction array, prior, TPR, TNR
    returns: posterior probability (updated prior)
    """

    for each_prediction in range(len(predictions)):

        if predictions[each_prediction] == 1:
            prior_ = 1 - prior
            bubbles_and_positive = prior * sensitivity
            no_bubbles_and_positive = prior_ * (1 - specificity)
            bubbles_when_positive = bubbles_and_positive /
                        (bubbles_and_positive + no_bubbles_and_positive)
            prior = bubbles_when_positive

        else:
            prior_ = 1 - prior
            no_bubbles_and_negative = prior_ * specificity
            bubbles_and_negative = prior * (1 - sensitivity)
            no_bubbles_when_negative = no_bubbles_and_negative /
                        (no_bubbles_and_negative + bubbles_and_negative)
            prior = 1 - no_bubbles_when_negative

    return prior
```

Sample implementation of GridSearchCV to get the optimal hyperparameters (XGB) using sklearn.

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold

seed = 80
nfold = 4

# XGB tuning parameters
parameters = {'learning_rate': np.logspace(-2, -1, 5),
              'n_estimators': np.logspace(1.75, 2.75, 5).astype(int),
              'subsample': (0.7,0.8,0.9,1.0)}

# Grid Search cross validation
cv = GridSearchCV(XGBClassifier(seed=seed), parameters,
cv = StratifiedKFold(n_splits=nfold, random_state=5), scoring='accuracy')
cv.fit(X_train, y_train)

# Train the model with the optimal parameters
model = XGBClassifier(n_estimators=cv.best_params_['n_estimators'],
seed=85, learning_rate=cv.best_params_['learning_rate'],
subsample = cv.best_params_['subsample'])
model.fit(X_train, y_train)

# Predict on the test data.
y_pred = model.predict(X_test)
```

C List of Acronyms

AE	Acoustic Emission
ANN	Artificial Neural Networks
AUC	Area under the Curve
BI	Bayesian Inference
BI	Business Intelligence
CART	Classification and Regression Trees
CBO	Cluster Based Oversampling
CF	Crest Factor
CM	Condition Monitoring
CM	Confusion Matrix
CPU	Central Processing Unit
CV	Cross Validation
DAQ	Data Acquisition System
DFT	Discrete Fourier Transform
EMG	Electromyography
FDI	Fault Detection and Isolation
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FN	False Negative
FP	False Positive
GNB	Gaussian Naive Bayes
GPU	Graphics Processing Unit
IoT	Internet of Things
IQR	Inter-quartile Range
KNN	K-Nearest Neighbour
ML	Machine Learning
MLP	Multi-layer Perceptrons
NPW	Negative Pressure Wave
P2P	Peak to Peak
PDF	Probability Density Function
PMF	Probability Mass Function
QLF	Quadratic Loss Function
RBF	Radial Basis Function
RF	Random Forest
RMS	Root Mean Square
ROC	Receiver Operating Characteristic
RVM	Relevance Vector Machine
SNR	Signal to Noise Ratio
SQL	Structured Query Language
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
WDN	Water Distribution Networks
WSN	Wireless Sensor Networks
XGB	Extreme Gradient Boosting