University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Data Science	Spring/ Autumn semester, 20.21. Open / ~~Confidential~~
Author: Fadwa Maatug	
Programme coordinator: Tomasz Wiktorski Supervisor(s): Professor Reggie Davidrajuh, Rituka Jaiswal	
Title of master's thesis: Anomaly Detection of Smart Meter Data	
Credits: 30	
Keywords: Power consumption, Anomaly detection, FB Prophet, Imbalanced classification, Machine learning.	Number of pages: ....87.......... + supplemental material/other: .... Stavanger, ...27/07/2021...... date/year

University
of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Anomaly Detection of Smart Meter Data

Master's Thesis in Computer Science
by

Fadwa Maatug

Supervisors

Reggie Davidrajuh
Rituka Jaiswal

Spring, 2021

“People are capable at any time in their lives, of doing what they dream of”

Paulo Coelho

Abstract

Presently, households and buildings use almost one-third of total energy consumption among all the power consumption sources. This trend is continuing to rise as more and more buildings install smart meter sensors and connect to the Smart Grid. Smart Grid uses sensors and ICT technologies to achieve an uninterrupted power supply and minimize power wastage. Abnormalities in sensors and faults lead to power wastage. Along with that studying the consumption pattern of a building can lead to a substantial reduction in power wastage which can save millions of dollars. According to studies, 20% of energy consumed by buildings are wasted due to the above factors. In this work, we propose an anomaly detection approach for detecting anomalies in the power consumption of smart meter data from an open dataset of 10 houses from Ausgrid Corporation Australia.

Since the power consumption may be affected by various factors such as weather conditions during the year, it was necessary to search for a way to discover the anomalies, considering seasonal periods such as weather seasons, day/night and holidays. Consequently, the first part of this thesis is to identify the outliers and obtain data with labels (normal or anomalous). We use Facebook prophet algorithm along with power consumption domain knowledge to detect anomalies for two years of half-hour sampled data.

After generating the dataset with anomaly labels, we proposed a method to classify future power consumptions as anomalous or normal. We use four different approaches using machine learning for classifying anomalies. We also measure the run-time of different classification algorithms. We are able to achieve a G-mean score of 97 per cent.

Acknowledgements

First and foremost, I am grateful to **Allah** the Most Merciful for allowing me to fulfil my goal of finishing my studies and answering my prayers.

I would like to express my sincere thanks to my supervisors, **Professor Reggie Davidrajuh** and **Rituka Jaiswal**, for their faithful encouragement, exceptional guidance, and comprehensive support through this dissertation. Their expertise in thesis research and their insights and suggestions aided me in improving my academic knowledge and overcoming the challenges I faced.

My heartfelt gratitude to my **husband** and my **kids** for their incredible support during my learning; this journey could not be possible without them. I thank my husband for his reassurance through my rough times. I am eternally thankful to my dear kids for their constant love and patience with my busyness and absence.

Last but most certainly not least, I am grateful to have wonderful **parents** who believed in me and motivated me to pursue my dreams. I am thankful for their encouragement to prove and improve myself in every aspect of my life.

Contents

Abbreviations

ICT	**I**nformation and **C**ommunications **T**echnology
OCC	**O**ne **C**lass **C**lassifier
ML	**M**achine **L**earning
GAM	**G**eneralised **A**dditive **M**odel
SMOTE	**S**ynthetic **M**inority **O**versampling **TE**chnique
ENN	**E**dited **N**earest **N**eighbours
MLE	**M**aximum **Likelihood** **E**stimation
LR	**L**ogistic **R**egression
DBSCAN	**D**ensity-**B**ased **S**patial **C**lustering of **A**pplication with **N**oise
RF	**R**andom **F**orest
OOB	**O**ut-**O**f-**B**ag
iForest	**i**solation **F**orest
iTree	**i**solation **T**ree
BTS	**B**inary **S**earch **T**ree
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
FPR	**F**alse **P**ositive **R**ate
TPR	**T**rue **P**ositive **R**ate
P	**P**recision
R	**R**ecall
ROC	**R**eceiver **O**perating **C**haracterictic
AUC	**A**rea **U**nder the **C**urve
GG	**G**ross **G**eneration

GC	**G**eneral **C**onsumption
CL	**C**ontrolled **L**oad
RMSE	The **R**oot **M**ean **S**quared **E**rror
TPE	**T**ree **P**arzen **E**stimator
KNN	**K** **N**earest **N**eighbour
RAM	**R**andom **A**ccess **M**emory

Chapter 1

Introduction

1.1 Background and Motivation

The recent smart grid is an enhanced electrical network infrastructure that incorporates advanced metering and information communication technologies in order to increase the efficiency, reliability, and security of the power grid. Additionally, it enables the collection, transmission, and storage of real-time data of energy consumption [1]. As a matter of fact, The smart grid is crucial for economic growth, energy structure adjustment, and climate change adaptation, all of which can lead to energy savings and pollution decreases [2].

Preventing and detecting power losses has become more necessary in today's global power system. Each year, the economy losses total hundreds of millions caused by illegal electricity use by customers in various countries' power industries [3]. With the growth and popularity of Internet of things, demand and supply complexity, monitoring and forecasting of power consumption is critical for power companies in terms of power generation, scheduling and dispatching. It benefits power consumers by allowing them to enhance their usage schedules and thus reduce their costs. Additionally, power suppliers can detect abnormal meter readings caused by unforeseen meter failures, intentional meter manipulations, or users' unusual consumption behaviors [1, 3, 4].

Discovering unusual meter measurements and unexpected customers habits is known as Anomaly detection or outlier detection. Anomaly detection is the process of identifying relatively infrequent events, and it has been widely applied in a variety of applications, such as fraud detection and fault detection in safety-critical systems. Anomaly detection can be applied to smart meter data to assist power consumers in identifying abnormal activities, such as forgetting to turn off irons after use, wasteful appliances, or over-lighting. Anomaly feedback can alert customers to reduce their energy consumption or replace inefficient appliances. Additionally, anomaly detection can assist suppliers in identifying energy leakage and theft, as well as unnoticed meter faults. For example, when daylight savings time begins, the meter may fail to record any consumption for that hour; similarly, when daylight savings time ends, and the clock is turned back, the meter may over-write the previous hour's data. Further, power utilities can establish a baseline for more precise demand-response programs for their clients [5].

1.2 Approach and Contributions

Since smart meters are digital devices capable of recording power usage at hourly or lesser intervals, they could still record consumer measurements in real-time or near real-time, enabling the monitoring of unusual activities or consumption patterns instantaneously [5]. This led us to develop a method for detecting abnormalities in historical consumption data and identifying the oddity within the consumption patterns using statistical approaches such as confidence interval estimation. Subsequently, we employ the most appropriate classification algorithm to ascertain if the future power consumption data is anomalous or normal.

The study consists of two phases in this thesis. In the first, we used a **Prophet** method, which analyzes and forecasts highly seasonal time-series data, producing predicted values and confidence intervals for trend changes. **Prophet**, is a Facebook-developed generalized additive model (GAM) that is mainly used to analyze time-series data and excels at forecasting highly periodic data [6]. Identifying the anomalies depends on the **Prophet** model's confidence intervals, which provides the regular interval of power consumption. Any data point that lies outside the interval is considered an outlier. Further, while zero energy consumption is illogical, we classify all zero values in the data as anomalous readings.

After classifying the data into two categories (positive class indicates anomaly, negative class indicates normal), we ended up with an unbalanced dataset. Thus, the second phase of our study includes experimenting with four alternative models: ***Cost-sensitive Logistic Regression***, ***Clustering using DBSCAN***, ***Ensemble Random Forest***, and ***One-class Isolation Forest***. Moreover, evaluating their performance using *Sensitivity*, *Specificity*, *G-mean*, and *F-score* measures emphasises minority class.

The contribution of this thesis is to offer a statistically-based anomaly detection methodology based on past consumption patterns of consumers. Additionally, to suggest the most appropriate classification approach for imbalanced data amongst the available strategies for predicting and classifying future data and assessing the model using the most suitable metrics.

1.3 Outline

This thesis is divided into the following chapters: Chapter 2 will introduce previously published works relevant to the thesis. Chapter 3 will demonstrate the principal methodologies of the thesis approach. Then the dataset structure will be presented in Chapter 4, as well as the approach implementation and experiments design. Chapter 5 will report and discuss the outcomes of the experiments as well as future suggestions. Finally, Chapter 6 will contain the conclusions to the thesis work.

Chapter 2

Related Work

Monitoring and detecting anomalies are essential tasks on time-series data of power consumption. Liu and Nielsen [5] proposed a model, which first applied a mixture of supervised learning algorithm called *Periodic Auto-Regression with eXogenous variables PARX* and *Gaussian statistical distribution* to detect the anomalies on historical consumption data. Second, they employed the *Lambda architecture*, which has three layers, Spark streaming as the speed layer technology, Hive as the batch layer technology and PostgreSQL as the serving layer technology, to detect anomalies in time and achieve much accuracy. Even though their results have validated the proposed system's effectiveness, their suggested method could not detect missing values, negative energy usage, and device faults. Fathnia et al.[7] provided a method that combined the *Local Outlier Factor LOF* index and *Ordering Points To Identify the Clustering Structure OPTICS* density-based algorithm to detect the unusual nature of the data. Their strategy recognised outliers as transmitted faults in the smart meter data to the control center. They reasoned that cyberattacks caused these failures. While the research findings demonstrated the efficiency of the proposed technique, they have not considered unnatural users' activities or inefficient appliances.

In the same direction, Zhang et al. [3] introduced an adaptive method to detect the abnormalities in smart meter data. They labelled the data set using the *Gaussian Mixture Model Linear Discriminant Analysis GMM-LDA* algorithm, which clusters some data sets to obtain the optimal feature representation of normal and abnormal patterns. Subsequently, they used the *Practical Swarm Optimization Support Vector Machine PSO-SVM* classifier to learn from the labelled data and predict future unlabelled data. While in [8], the authors experimented with two approaches for detecting oddities in real and synthetic datasets. They started with a statistical technique, which uses *Standard Deviation* to identify extremes. The second technique utilised the *K-Nearest Neighbor*

KNN clustering algorithm to distinguish anomalous from normal data using the point-to-point distance measure. Research work by Janetzko et al. [9] defined aberrant behaviour in the principal scenario of utilisation as a variation from the expected daily pattern. They introduced two methods that can be adapted to the periodicity of the energy consumption data. The first approach estimated the error rate using weighted prediction, where the more substantial influence of current measurements than older ones. The latter approach used similarity-based anomaly detection after transforming the daily pattern into the frequency domain by Fourier transformation. Additionally, they provided a range of time series visualisation techniques for resulting anomaly scores that aid in analysing and comprehending energy consumption behaviour. Nevertheless, a significant drawback of all three previous papers is that they have not considered the effect of exogenous factors, such as weather conditions, on the power demand.

In light of the shortcomings of the mentioned prior works, we presented a technique to detect a wide range of anomalies considering the multi-period seasonality that correlated to the weather conditions and human behaviours.

Chapter 3

Methodology Background and Theoretical Structure

The necessary theory and methodology background are introduced in this chapter, including the Prophet algorithm and preprocessing techniques. In addition to four machine learning models such as Cost-Sensitive Logistic Regression, DBSCAN, Ensemble Random Forest and One-Class Isolation Forest. In the end, the chapter provides a brief knowledge of model evaluation metrics.

3.1 Introduction

Before we begin, it is necessary to establish some fundamental definitions. As a start, let us define Machine Learning (ML). It is a set of fundamental mathematical techniques used for extracting significant characteristics from the data. The learning can be made by utilising unique meaningful patterns formed from the data and generating predictions. There are several types of machine learning, including semi-supervised learning, reinforcement learning, supervised learning, and unsupervised learning. We are particularly interested in the latter two. Supervised data mining algorithms are presented with labelled data sets, where classification may be used if the labels are categorical; otherwise, a regression can be applied to optimise the model for the labelled data provided. In contrast, for unlabeled data, an unsupervised learning algorithm must be employed to discover patterns in the data in a principled manner to determine how to cluster and categorise new data [10].

In this thesis, a detection method is proposed to find the anomalies in consumers' history consumption via the unsupervised learning and statistical algorithm, which resulted in labelled data with an imbalanced distribution of the known classes. Imbalanced classification is the process of classifying this sort of data. In the two-class (binary) imbalanced classification problem, the minority (underrepresented) group is typically referred to as the positive class, while the dominant group is the negative class [11].

There are four major categories of approaches that may accurately identify the minority class according to [11–13]:

- Algorithm-Level Approaches: These techniques try to change the classifier learning procedure without affecting the data distributions. It is focused on a single classifier type, making it more adaptable to different types of imbalanced datasets. This requires an in-depth study of the chosen strategy to ascertain which precise mechanism may be accountable for forming a bias against the majority class. This category includes One-Class Classifier methods OCC, which can be used for binary classification problems with a severely skewed class distribution. These methods can be trained on the data-set's majority class and treat the minority class as outliers.
- Data-Level (Preprocessing) Approaches: These approaches employ a variety of procedures to modify and rebalance the distribution of classes. Rebalancing datasets improves classification tasks significantly when compared to unbalanced sets.
- Cost-sensitive learning Approaches: They integrate both Data and Algorithm levels, where costs added to data samples, and the learning method is modified to admit these costs. Higher misclassification costs are assumed for the minority class, aiming to reduce the entire cost error for both classes.
- Ensemble-based Approaches: They are frequently combined with one of the above mentioned approaches, particularly data-level and cost-sensitive learning (the cost minimisation is supervised via the ensemble algorithm). The ensemble classifier usually uses boosting technique, where the samples trained serially with the training's weights adjusted adaptively during the training process according to the preceding classifier.

We shall employ all of the techniques' types stated above in our thesis and evaluate them using the particular metrics discussed later in this chapter.

3.2 The Prophet Method

The Prophet is an Generalize Additive Model GAM that Facebook has as open-source. Additionally, Facebook modified Prophet parameters in R and Python and adopted it as its application. Prophet's processing capability is exceptional for forecasting data with high seasonal effects and long-term non-stationary rends [14]. Besides, the Prophet is based on the Bayesian model's curve fitting approach. It is attempting to fit different linear and non-linear time functions as elements. Further, Prophet forms seasonal components as additive elements, same strategy as exponential smoothing. It features simple-to-understand parameters and does not need much time-series data to make a prediction. Moreover, the Prophet technique is capable of managing both stationary and non-stationary components. Not only it can handles the time-series data affected by solid seasonal factors, but it also performs well in the presence of missing data, trend variance, and outlier detection. Additionally, it accounts for scheduled breaks or holidays in continuous data [15, 16].

The Prophet core is to work on as an additive regression model that accommodates three components as indicated by the following mathematical notation and as mentioned in [17–19].

$$y(t) = g(t) + s(t) + h(t) + \epsilon(t) \tag{3.1}$$

where,

- $g(t)$ is the trend function that represents the linear or logistic non-periodic changes in power consumption readings obtained at evenly time-space intervals.
- $s(t)$ is a seasonal component formed using Fourier series, and it captures the historical data periodic changes, which can be daily, weekly, monthly or yearly seasonality. In this study, we focused on daily, weekdays, weekends, yearly seasonality, so the Prophet model regression equation will be as following [20]:

$$y(t) = g(t) + s(t)_{daily} + s(t)_{weekdays} + s(t)_{weekends} + s(t)_{yearly} + h(t) + \epsilon(t) \tag{3.2}$$

- $h(t)$ represents the holidays during the year and can be provided by the user or unusual expected times that can happen irregularly.
- $\epsilon(t)$ indicates an independent error term, which assumed to be normally distributed.

A drawback of the Prophet that it requires two input columns with the name ds and y, where the first represents the date and the second represents, in our case, the power consumption. Afterwards, the model can be fitted on the train data and finally, the forecasting is performed on the data. Additionally, the prediction in Prophet not only returns the estimated values but also returns uncertainty intervals for each component, such as $\hat{y}_{lower}$ and $\hat{y}_{upper}$ for the forecast $\hat{y}$. These are quantiles of the posterior predictive distribution, and specifying which quantiles to employ can be done via the Prophet's library parameter `interval_width`. Adjusting the `interval_width` parameter will only modify the uncertainty interval and will not affect the prediction values [21].

When the training data is fitted to the Prophet model, a collection of datasets will be generated from the likelihood and priors of the model. This collection will form the posterior predictive distribution, which illustrating how predicted data may appear considering all the average frequency and magnitude of history data trend changes [21, 22]. In this project, the posterior predictive distribution is employed to show how the normal behaviour of the data should be. We considered an uncertainty interval by taking the *99-th* quantile of the posterior predictive distribution, which has a 99% chance of containing the parameter's actual value [23]; Besides, any value that lies outside the interval would identify as an outlier. We define the uncertainty interval $\boldsymbol{I}$ as:

$$I = [\hat{y}_{lower}, \hat{y}_{upper}] \tag{3.3}$$

We established the following decision rule for an original data sample y [24]:

$$\begin{cases} \text{y is normal, if } y \in I \\ \text{y is anomaly, if } y \notin I \end{cases} \tag{3.4}$$

3.3 Data Sampling

One of the most popular imbalanced classification solutions is to change the structure of the training dataset, which is referred to as Data sampling techniques. They are simple to comprehend and apply without requiring any adaptation to account for the observational imbalance. After implementing them on the training dataset, the ML algorithms options will increase, including the algorithm developed for balanced classification [12].

Data sampling techniques can be divided into three groups [11, 12]:

- **Undersampling methods** are removing majority class instances to form a subset of the original dataset. Edited Nearest Neighbour Rule (ENN) is an example of undersampling methods. ENN is a data cleaning-based technique that eliminates majority class instances that differ from two of its three Nearest Neighbours.
- **Oversampling methods** replicate some instances or generate new instances from existing ones that belong to the minority class to produce a comprehensive of

the original dataset. One of the Common oversampling approaches is Synthetic Minority Oversampling Technique (SMOTE). It selects instances that lie nearby in the feature space to create new minority class examples. The over generalisation problem is a limitation of oversampling due to the way it generates synthetic data samples without regard for close instances, which increases the likelihood of class overlapping.

- **Hybrid methods** combine both sampling approaches. For instance, (SMOTE + ENN) is a widely used method. SMOTE and ENN combination methods apply SOMTE first. It then eliminates instances from both classes using ENN, aiming to reduce noisy points along with the classes boundaries, which improves the performance of the classifiers fitted to the modified dataset.

3.4 Cost-Sensitive Logistic Regression Algorithm

Cost-Sensitive learning is a recent ML approach that trains classifiers to distinguish the various costs associated with several classification errors. Further, it may be classified into **direct** and **indirect** cost-sensitive techniques. A direct cost-sensitive learning algorithm is built by including various misclassification costs into the training process directly. In contrast, the indirect method may convert a classifier to be cost-sensitive by preprocessing the training data through data sampling methods [25].

Logistic Regression LR is a linear model with the optimal coefficients determined by maximizing the log-like function. LR estimates a positive class's posterior probabilities in a particular context of binary classification [25, 26]. High interpretability and low time complexity are advantages of LR. Penalising the misclassification costs differently while training the model within the log-likelihood objective function can moderate the bias on the imbalanced data [26].

The predicted probability of the positive class for a given sample X_i is estimated as $P(y = 1|X_i)$, and for the negative as $P(y = 0|X_i) = 1 - P(y = 1|X_i)$ [25]. Besides, The values of input independent variables (i.e. $x_{j0}, ..., x_{jm}$) are linearly combined in LR [26], as defined in the following equation [25], then transformed by a logistic (sigmoid) function as defined in equation (3.6) [25]:

$$g(x, \beta) = \beta_0 + \sum_{j=1}^{m} \beta_j x_j \qquad j = 1...m \tag{3.5}$$

where β is the estimated coefficients of the independent variables, m is the total number of explanatory variables.

$$p_i = P(y = 1|X_i) = f(g(x, \beta)) = \frac{1}{1 + e^{-}g(x, \beta)} \qquad i = 1...n \tag{3.6}$$

where n is the total number of observations, and the value of p_i is restricted between 0 and 1 and it refers to the hypothesis of i given the parameters Θ. p_i is the probability of the sample predicted to be 1 when the true value is 1; the difference between the estimated and truth will be smaller as the p_i value is bigger. While $1-p_i$ is the probability that the sample is predicted to be 0 when the actual value is 0, as $1 - p_i$ is closer to 1, the difference between the predicted value and the true value will be smaller. In addition, to assure that a consistent estimate with an actual value, the formed LR model is determined using the maximum likelihood estimation MLE, which assume that each data sample is independent. β is is achieved by maximizing $L(x, \beta)$ in equation (3.7) to ensures that the difference between the predicted and the actual value is the smallest [25]. The below equations illustrates the standard MLE function according to [25, 27] respectively.

$$L(x, \beta) = \prod_{i=1}^{n} f(g(x, \beta))^{y_i}.(1 - f(g(x, \beta))^{(1-y_i)}) \tag{3.7}$$

$$lnL(\beta|y) = \sum y_i ln(\pi_i) + \sum (1 - y_i) ln(1 - \pi_i) \tag{3.8}$$

Nevertheless, the Cost-Sensitive Logistic Regression is employed the weighted MLE function, which adds a positive class Weight W_1 and a negative class Weight W_0 to punish the misclassifications of anomaly and normal differently. The estimated population proportion of events τ and the sample proportion of events $\bar{y}$ defines the classes weights [27]. The weighted MLE equation is defined in (3.9) [27].

$$lnL_W(\beta|y) = W_1 \sum y_i ln(\pi_i) + W_0 \sum (1 - y_i) ln(1 - \pi_i) \tag{3.9}$$

Where,

$$\pi_i = \frac{1}{1 + e - (\beta_0 + \beta_1 x_1 + ... + \beta_m x_m)} \qquad , W_1 = \frac{\tau}{\bar{y}} \quad , \quad W_0 = \frac{1 - \tau}{1 - \bar{y}} \tag{3.10}$$

3.5 Clustering using DBSCAN Algorithm

When the data has no defined label, such as "anomaly" or "not-anomaly", the unsupervised detection technique is utilised. Clustering is an unsupervised technique for identifying patterns in data groups across a wide range of data, and studies have demonstrated that anomaly detection may be done using clustering [28].The anomaly detection methods can be divided into two approaches, which are *statistical approaches* and *distance-based approaches*. The first type seeks to create a statistical model of the data and identify data that does not fit the model as an outlier. The latter type considers the distance between the data to identify the outliers, which has a greater distance than the predefined distance [29]. This section of the study concentrates on detecting outliers using the **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** method to define normal and anomalous data groups and evaluate the model's labels across the Prophet labels.

DBSCAN is one of the most potent algorithms on mass and dense datasets for finding anomalies, where it will create membership points labelled as cluster members or as an outlier that do not fit any clusters. DBSCAN requires two user-defined parameters to identify a cluster: the maximum neighbourhood distance ***epsilon*** ϵ and the minimum number of the points within the cluster ***minpts*** [29]. Starting with a random data point $\vec{p}$, DBSCAN locates all the neighbour points within radius epsilon distance from the point $\vec{p}$, based on the following equation [30].

$$N_\epsilon(\vec{p}) = \{\vec{q} \in D | dist(\vec{p}, \vec{q}) \leq \epsilon\} \tag{3.11}$$

The distance between $\vec{p}$ and $\vec{q}$ is calculated based on Euclidean distance [30]:

$$dist(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{m} (p_i - q_i)^2} \qquad i = 1...m \tag{3.12}$$

Where $\vec{p}$ and $\vec{q}$ are a vectors points with m features. The neighbouring points of $\vec{p}$ are called a *cluster* if they are more than or equal ***minpts*** [29]. DBSCAN is continued until labelled all the data points into one of the three categories: *Core points,* which are the points within a cluster. *Border points*, which are not core points but are their neighbours. Last, *Outlier points*, which are not core points or border points and not members of any of the clusters [28, 29]. In other words, a *cluster* can be defined as a set of members containing core points circled by border points [28].

3.6 Ensemble Random Forest Algorithm

Ensemble learning is a consensus technique in which is used to aggregate the predictions in order to enhance the classifier model's performance. It is intended to rectify the errors of its classifiers members. Hence, each classifier within the ensemble might make various errors in different training sample instances as a strategy. The combination of these several classifiers can minimize the overall error [31]; some of the ensemble methods are Bagging and boosting [32].

The **Random Forest RF** classification method is an ensemble of many decision tree classifiers that uses bagging, i.e., bootstrap aggregation [32], and the outcome of RF is a combination of all the ensemble trees' predictions [31]. In other words, each tree comprises several bootstrap samples created by original samples referred to as bagging, which created multiple training data values by randomly resampling the original dataset with replacement. In order to avoid tree correlation, bagging generates a variety of trees completing multiple training data subsets [33]. Furthermore, a classification tree separates data recursively into subgroups without considering the underlying connections between predictor factors and response variables [32]. The decision trees will have low bias and high variance when they grow deeply, leading to irregular patterns learning and overfit training. Random forests give improvement over just bagged trees because they decorrelate the trees while building the RF [31]. Specifically, The training data is sampled to create two partitions; an *In-Bag* division constructs the tree; a smaller division named *out-of-bag (OOB)* tests and evaluates the built tree's performance [32]; the *OOB* error is an unbiased estimation of the generalisation error that provides information about significant factors [33].

The RF algorithm's fundamental ideas include that each sample obtained from the dataset for the training process can be reused for another training tree operation. Additionally, the features utilized in each tree during the training phase are a subset of the features in the dataset [34]. As discussed in [32], the following steps illustrate how a decision tree in the RF is constructed:

1. The number of trees m is determined to be used with the following criteria:
    - A bootstrap sample of size n is produced, and an S_n sample is chosen for tree growth.
    - To create a tree at each node, f features are randomly chosen and utilised to determine the optimal split.
    - The tree is allowed to develop to its full potential without being pruned.
2. The maximum number of features k that will be used to split each node is determined, and it should be less than the total number of input data features. Throughout the process of forest development, the subset of features k is maintained constant.

3. A majority voting system categorises a sample X by evaluating votes from each tree in the forest.

For better classification results, a higher value of m might be chosen, while for k value, it is recommended to use square root or logarithm of the total number of features in the dataset [34].

As stated in [33], using the RF classifier has several advantages, as it has a low computational cost and compelling performance on massive datasets. Besides, it can perform many input features without feature exclusion. Further, RF can determine the essential variables for classification, establish a correlation between variables and classification, and avoid overfitting.

3.7 One-Class Isolation Forest Algorithm

Unsupervised anomaly detection techniques are based on two essential assumptions. To begin, the vast majority of events are normal, with just a small number being aberrant. Second, the outlier deviates statistically from the norm [35]. **Isolation Forest (iForest)** is an unsupervised ensemble learning-based method for detecting anomalies. The approach generates anomalous scores by learning the data set's tree structure, eliminating the need to calculate costly distance or density measurements. By partitioning the sample space, it creates a forest with a limited height; anomalies are more possible to be located at a shallow depth in a tree than non-anomalous values. As with random forests, iForest is composed of a vast number of isolated trees (iTree). Therefore, iForest may split into two stages. Firstly, create iTrees and establish a forest. Secondly, the anomaly score for the identified samples is computed [36–38].

The iForest algorithm chooses a random feature from a random subset (or the whole set) of all features during each iteration of tree construction. Subsequently, using a random split value within the range of this feature's minimum and maximum values, a subset of the current node samples is split into two subgroups. Further, when each tree node includes exactly one sample (leaf node), the split is complete. The ***path length*** from the root node to the selected leaf node is the number of splittings required to isolate this sample, and it represents a *measure of normality.* This random partitioning results in noticeably shorter pathways for anomalies because fewer anomalies result in fewer partitions – shorter paths in a tree structure - and instances with distinct attribute values are more likely to be segregated during early partitioning. Therefore, when iTrees produces a shorter path length for particular samples, they are likely to be anomalies [39, 40].

According to [36], iTree's production process is as follows: for a given sample with Q-dimension distribution of a data sets $D = x_1, ..., x_k, ..., x_\psi$ which has ψ examples.

1. Choose a feature $a_i \in Q$ at random.
2. Choose a split value from (min,max) range values of $a_i \rightarrow V^{'}$.
3. According to the feature V_{aik}, for each sample x_k, the V_{aik} that is smaller than $V^{'}$ is categorized as left child node set, and the remaining as right child node set.
4. Recursively create the left and right child sets until: the entering data set D has just one record or all data in D has the same value; the tree reaches the l-threshold height.

According to their path lengths or anomaly scores, sorting data points is the way to detect anomalies in this method. The path length $h(x)$ of a new example x is determined as the total number of edges traversed by x from the root node to its belonging leaf node. The average height grows in the order of $\log(\psi)$. Since iTree's structure is similar to that of a Binary Search Tree (BTS), the estimated average path length in iTrees is [36]:

$$c(\psi) = 2H(\psi - 1) - \frac{2(\psi - 1)}{\psi} \tag{3.13}$$

where $H(i)$ is the harmonic number and it can be estimated as [36]:

$$H(i) = \ln i + 0.5772156649 \tag{3.14}$$

where the constant introduced is Euler's constant. Further, the anomaly score s of an sample x is defined as [36]:

$$s(x, \psi) = 2^{\frac{-E(h(x))}{c(\psi)}} \tag{3.15}$$

where $E(h(x))$ is the expected of path length $h(x)$ from a collection of iTrees [36]. As discussed in [40] in equation 3.15:

- when $E(h(x)) \rightarrow c(\psi), \quad s \rightarrow 0.5$
- when $E(h(x)) \rightarrow 0, \quad s \rightarrow 1$
- and when $E(h(x)) \rightarrow \psi - 1, \quad s \rightarrow 0$

We are capable of making the appropriate determination using the anomaly score s, since s is monotonic to $h(x)$ and applied the conditions where $s \in [0, 1]$ for $0 < h(x) \leq \psi - 1$ [40]:

- Instances identified as anomalies when s very close to 1, or 1.
- Instances regarded as normal when s is much smaller than 0.5, or 0.
- The whole sample does not have any distinct anomaly when s for all the instances ≈ 0.5.

3.8 Model Evaluation

The classification algorithm's performance is often assessed by comparing the predicted class labels to the real ones. The model is trained on the training set and then evaluated on the holdout test set. Data preprocessing and model selection are an exploration area that needs to be driven by the evaluation metric. Consequently, various models must be used in the experiments, and each experiment's outcomes should be quantified using a suitable metric. Besides, standard evaluation metrics, such as Accuracy (which is the percentage of accurately categorised samples), evaluate all classes' importance equally. Alternatively, in an imbalanced classification problem, the majority class is rated as less important than the minority class. Therefore, performance metrics concentrating on the minority class may be both important and problematic, as the minority class lacks the requisite samples for training an effective model [11, 12].

Before presenting the evaluation matrices that have been used, we need to understand the Confusion Matrix for binary classification tasks. The confusion matrix can provide more insight into the model performance and gives information about the correctly and incorrectly predicted labels [12]. Besides, it has four possibilities: True Negative TN, which means the truth labels are negative as the predicted labels; True Positive TP indicates that the truth and predicted labels belong to the positive class; False Positive FP shows the number of actual negative samples that predicted as positive labels; Last False Negative FN presents the number of positive data points predicted as negative labels. We focus while evaluating the best model on the latter case, False Negative, since it shows how many true anomalies are predicted as normal samples; And it must be as lower as possible.

The character of anomaly detection application demands a relatively high rate of accurate detection in the minority class while allowing for a small error rate in the majority class to achieve this [13].

	Negative Prediction	Positive Prediction
Negative Class	True Negative (TN)	False Positive (FP)
Positive Class	False Negative (FN)	True Positive (TP)

Table 3.1: Confusion Matrix of Binary Classification

The following paragraphs will present the matrices employed in our study as mentioned in [11–13, 41].

- **Sensitivity** indicates the accuracy with which the positive class was anticipated.[1]

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.16}$$

- **Specificity** indicates the accuracy with which the negative class was anticipated.

$$Specificity = \frac{TN}{TN + FP} \tag{3.17}$$

- **G-mean** combines Sensitivity and Specificity into a single score that takes both considerations into account.

$$G - mean = \sqrt{Sensitivity \times Specificity} \tag{3.18}$$

- **Precision** quantifies the proportion of instances allocated to the positive class that actually belong to it, and it is suitable when the objective is to minimise false positives.

$$P = \frac{TP}{TP + FP} \tag{3.19}$$

- **Recall** quantifies the proportion of the true positive samples that correctly detected by a classifier; And it is suitable when the objective is to minimise false negatives.

$$R = \frac{TP}{TP + FN} \tag{3.20}$$

- **F-measure** focuses on the analysis of positive classes. Its purpose is to examine the trade-offs between accuracy and coverage while categorizing positive examples. The measure does this by calculating a weighted harmonic mean between precision and recall, a high score of precision and recall will lead to a high F-measure value. The general β formulation of F-measure is shown in the next equation, where β is a parameter that manages the importance provided to each term. A common choice

[1]It's important to note that the same equation is used to compute ***sensitivity***, ***Recall R***, and ***True positive rate TPR***.

is setting $\beta = 1$, which balance the weight on precision and recall, leading to the F1 measure (F1-score), and when $\beta = 2$ is referred by F2 measure (F2-score), it gives less weight on precision, more weight on recall. Both measures are employed in the study.

$$F_\beta = (1 + \beta^2) \times \frac{P \times R}{(\beta^2 \times P) + R} \tag{3.21}$$

$$F_1 = 2 \times \frac{P \times R}{P + R} \tag{3.22}$$

$$F_2 = 5 \times \frac{P \times R}{(4 \times P) + R} \tag{3.23}$$

- **True Positive Rate TPR** It is the possibility that an actual positive will be detected.

$$TPR = \frac{TP}{TP + FN} \tag{3.24}$$

- **False Positive Rate FPR** It is the probability of receiving a positive label when the real value is negative.

$$FPR = \frac{FP}{FP + TN} \tag{3.25}$$

- **ROC Curves** The Receiver Operating Characteristic (ROC) curve is a graphical evaluation technique that widely used for summarising classifier performance over a range of true positive and false positive error rates trade-offs. Additionally, ROC demonstrates that the true positive rate cannot be increased without raising the false positive rate for any classifier. Furthermore, the Area Under the Curve (AUC) summarises a classifier's achievement into a single measure. It can evaluate a model performance as independent of the decision criterion and prior probability used; The higher the AUC score, the more accurate the classifier, and the optimal classifier would score one.

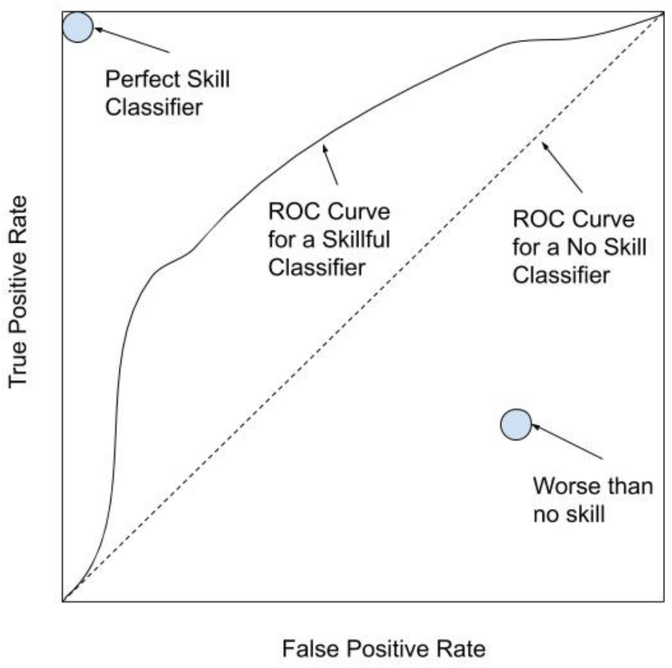


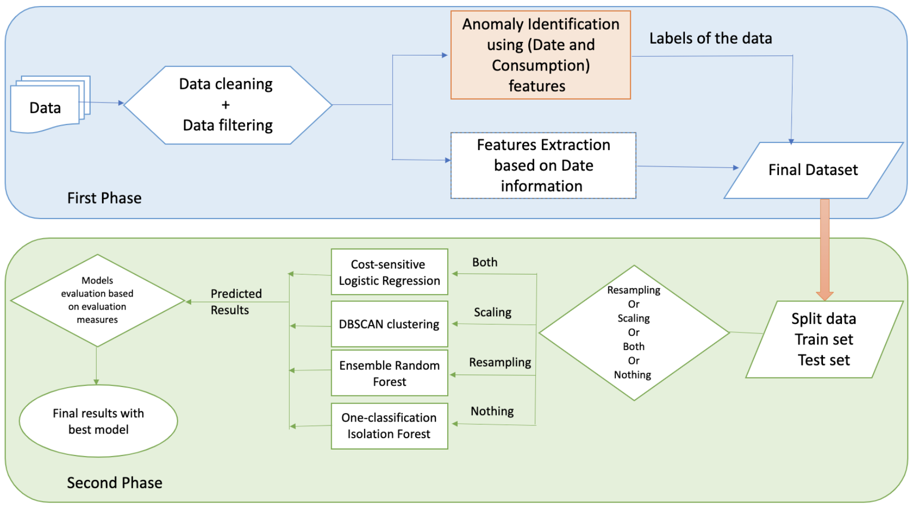
Figure 3.1: Illustration of a ROC Curve [12].

Chapter 4

Solution Approach And Implementation

4.1 Proposed Solution Introduction

The primary purpose of this study is to detect anomalies in power consumption data and obtain the best model to classify future events. This chapter outlines the steps necessary to accomplish this goal. Besides, it defines the utilised dataset and Python libraries used to fulfil the study's implementation. Further, we discuss the experiments that were conducted and the evaluation's implementation in this chapter.



Data
Data cleaning + Data filtering
Anomaly Identification using (Date and Consumption) features
Labels of the data
Features Extraction based on Date information
Final Dataset
First Phase
Split data Train set Test set
Resampling Or Scaling Or Both Or Nothing
Both
Scaling
Resampling
Nothing
Cost-sensitive Logistic Regression
DBSCAN clustering
Ensemble Random Forest
One-classification Isolation Forest
Predicted Results
Models evaluation based on evaluation measures
Final results with best model
Second Phase

Figure 4.1: The solution approach's workflow.

Figure 4.1 is a graphical presentation that illustrates the solution approach's workflow, divided into two stages. In brief, the first phase starts with collecting and analysing the data set, following with cleaning and filtering the data to be ready for identifying anomalies by using the *Facebook Prophet* algorithm. Besides, extraction some features based on *Date feature* information to use later with *ML* algorithms. The first phase's outcome is a data set that contains the extracted features and the *Prophet* model's labels. After this point, the second phase begins by splitting the dataset as an experimental set-up into two sets,*Training* and *Testing*. Following this, Data preprocessing is applied as needed, such as *data sampling* and *data scaling*, which is crucial for better performance with applied models. After that, we utilised four *ML* models: *Cost-Sensitive LR*, *DBSCAN*, *Ensemble RF*, and *OCC iForest*. The models' evaluation is implemented at the final step, describing the models' performance and identifying the best model.

4.2 Ausgrid Dataset

The data we used is available by an Australian energy provider (Ausgrid) in New South Wales (NSW), Australia. Ausgrid dataset was collected from 300 randomly selected solar customers in Ausgrid's energy network region charged on a residential tariff and installed a gross metered solar system between 1 July 2010 and 30 June 2013 registered at half-hour intervals. The Ausgrid corporation used three separate meter recording devices for three different categories (Gross Generation GG, General Consumption GC, Controlled Load CL). The first meter records the solar generation (GG) of the solar PV units placed at the rooftop of each household. At the same time, the second meter records the daily power consumption (GC) in (KWh) for each customer over 48 periods of the day; the period duration is a half-hour. While the last meter correlated with water heating is placed in some of the houses by offering a monetary incentive, these devices allow the provider to control customers' water heating system for managing the entire demand on the network [42]. Furthermore, Ausgrid company deleted from their dataset any personal information that may identify the customers to comply with clients' privacy protection. As a result, the dataset contains customer IDs ranging from 1 to 300 and the postcode associated with each consumer, which totals 100 different postcodes. In summary, for each line, the dataset file contains 54 columns. Starting with five description columns: Customer IDs, Postcode, Generator Capacity, Consumption Category and Date, followed by 48 columns of half-hour power meter data. In addition to the last column (Row Quality), which defined the type of recorded data as actual or estimated reading [43].

In this project, we used Ausgrid's dataset from 1 July 2011 – 30 June 2013, separated into two files (2011-2012 Solar home electricity data) and (2012-2013 Solar home electricity data) .

4.3 Phase 1

This stage explains the data preprocessing steps required for both (anomaly detection and anomaly classification) and how we identified the outliers in this study. Moreover, the data preprocessing illustrates the five significant tasks applied to the data. It starts with data cleaning by removing any missing values to enhance the data quality, following this, data filtering by choosing the most relative features. After that, the data is explored and visualised to interpret the data thoroughly. Next, the feature extraction task defines the predictive features based on date information, and feature transformation by converting data into a suitable format for ML models. However, the anomaly identification part determines the Prophet algorithm and the anomaly decision rules implementation and discusses the model's necessary parameters tuning.

4.3.1 Phase 1 - Data preprocessing

The Anaconda platform is utilised to implement the code via Python programming language, particularly the Jupyter notebook application. The Jupyter application is a Web-based interactive computing notebook environment used to edit and run Python code and human-readable documents while representing the data analysis [44]. Moreover, the Ausgrid datasets are combined in a zipped file, and each year's data is contained in a separate CSV file. We began by uploading the dataset files for two years, (2011-2012) and (2012-2013), with Pandas data frame to perform the data preprocessing. The following five sections illustrate the data preprocessing tasks applied to the datasets.

Data Cleaning

Data cleaning is a necessary step since the smart meter data may contain some missing values resulting from an error during the transportation or faults in smart meter records. Additionally, ML algorithms perform efficiently without the missing values that may affect parameters' estimation. To ensure the reliability of our work, we retained only the actual data and eliminated all Non-Actual values based on the Row Quality feature's categories:

- Row Quality is (Blank), indicates that row data represents the actual electricity measured by the meter during that half-hour [43].
- Row Quality is (NA), implies that the row data are estimations or substitutes for the power used or generated [43].

On the other hand, we held all zero numbers and classified them as outliers because zero power consumption is irrational and clearly indicates a mistake in the smart meter readings.

Data Filtering

The following are the primary reasons for filtering the dataset:

- The dataset contains two years of readings for 300 customers, which is an enormous amount of data for experimentation; as a result, we picked only ten customers to represent a small community and therefore minimise the amount of time required for experimentation. To accomplish this task, we analysed the Postcode feature and selected the area with the most customers. Since the selected customers had the same postcode, we removed the Postcode feature and limited the customers' IDs feature to ten.

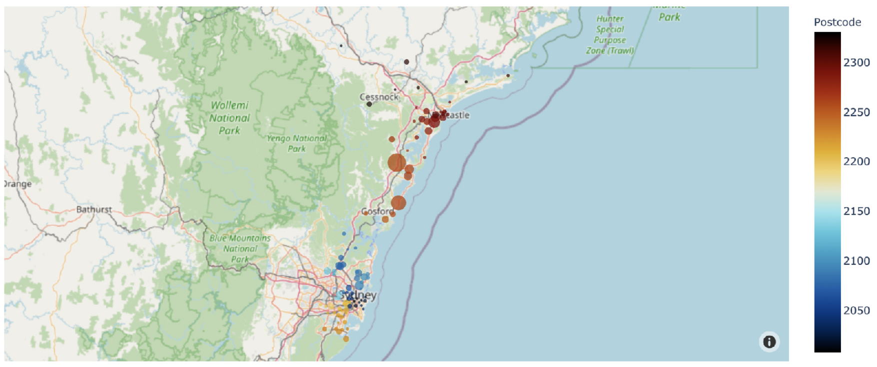


Figure 4.2: The distribution map of Ausgrid. The circles represent postcode regions covered in the 300-customer dataset.

Our files contain postcodes ranging from 2008 to 2330, which relate to New South Wales (NSW), Australia. Additionally, as seen in Figure 4.2, the largest area is located between Newcastle and Sydney; and its postcode value is 2259. Besides, The selected customers are the top ten customers intersected within both dataset files (2011-2012) and (2012-2013), the ten selected customers' IDs as shown in table 4.1.

7	29	30	64	155
160	184	202	206	215

Table 4.1: Selected Customers' IDs

- This study aims to detect the outliers in the residential consumption load of the households; therefore, we eliminated all the data features relevant to the power generation, including the Generator Capacity and all the generation (GG) readings. Similarly, we excluded all Controlled-Load (CL) readings as this service is not available in every house. Further, since they are measured with a separate meter device, the readings would have different patterns which could not be added to the target general consumption. As a result, we retained only the General Consumption (GC) readings, which represent the primary and the most significant part of the household's daily energy consumption.

- Due to the Prophet algorithm input's restriction, which requires only two features with specific names, We reshaped the dataset and combined each date with the 48 half-hour intervals as *Date_Time* feature, and we listed all the corresponding GC readings as *General Consumption* feature. The dataset was filtered and converted from 54 features into two per ten unique customer IDs over two years. We applied the Prophet algorithm on each customer independently as an individual experiment; because each customer has unique behaviour on daily consumption.

Data Exploration

To obtain a summary of the energy outlines of the customers, we started by aggregating the data annually using the summation method over a year for each customer. Figure 4.3 shows the overall households' consumption over two years. It indicates how the individuals' behaviour varies from one to another, as some of the houses have more power demand than others. Additionally, energy usage differs by year, such as in 2012, where it has the most consumption, these changes may have occurred as a consequence of external factors, for example, weather conditions.

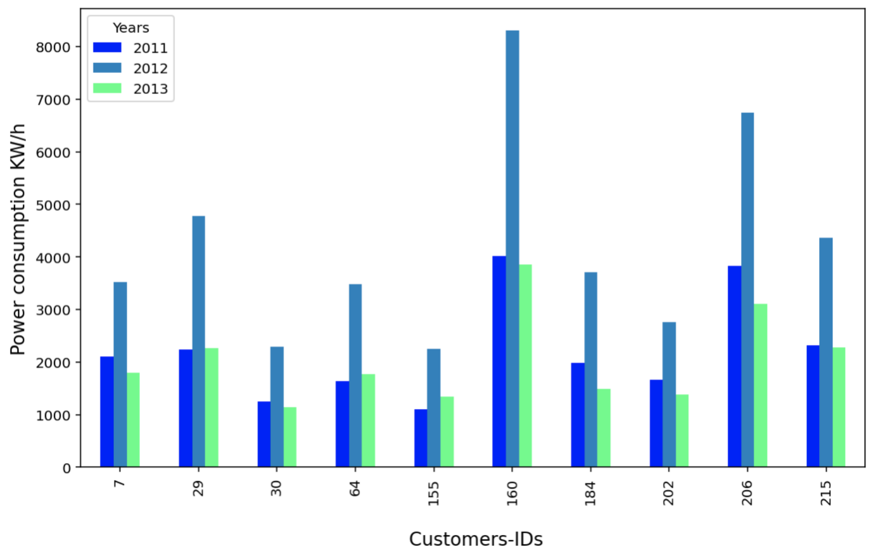


Figure 4.3: Total power consumption over two years for the selected customers.

Furthermore, we combined all the consumption historical data of our selected consumers into a single aggregated consumer. The energy form of this aggregated consumer was resampled on a seasonal basis (three months). Bearing in mind, that the seasons in Australia defined differently. For instance, the Winter period includes (June - July - August), the Spring period includes (September - October - November), the Summer period includes (December - January - February), and the Autumn period includes (March - April - May). Figure 4.4 shows the peak in total power usage corresponding to winter/2012, in addition to summer/2012, which have more power demand than spring/2012 and autumn/2012 due to weather requirements, as people tend to use air conditioning in the summer and heating in the winter. However, winter/2011 recorded the lowest consumption value, due to the warm winter.

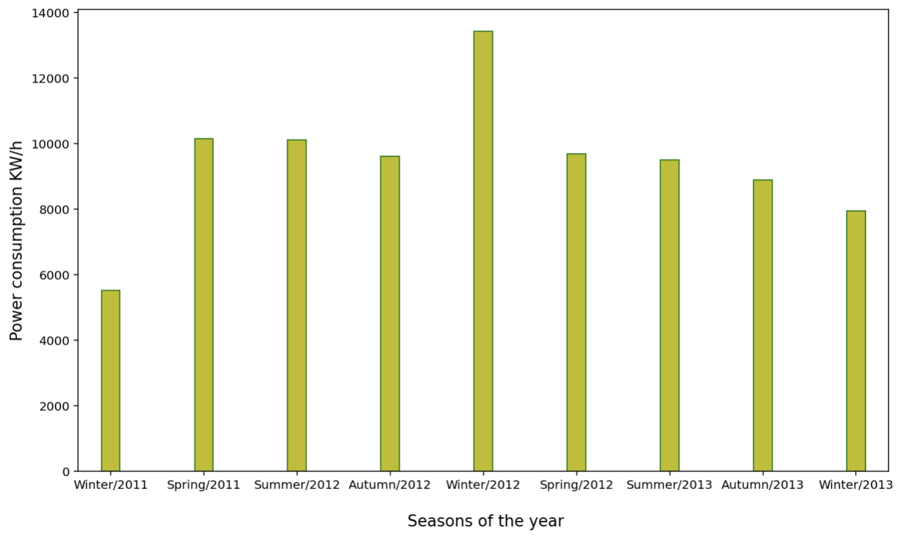


Figure 4.4: Total seasonal power consumption over two years aggregated across all customers.

Similarly, the aggregated consumer was grouped on a day-of-week and a monthly basis, which enabled us to compute a daily power form for a total aggregated customer over 12 months. Figure 4.5 determines that the highest energy demand across all selected customers was in July (winter season), while the lowest demands were in September and October (autumn), which confirm the conclusions made in figure 4.4. Besides, it shows how consumers prefer staying home during the weekends because the consumption is higher than on weekdays. In contrast, Thursdays have the lowest power demand amongst all weekdays.

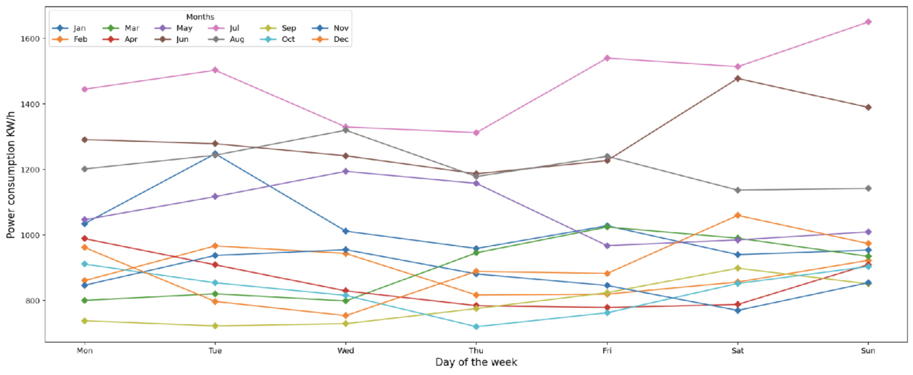


Figure 4.5: Total power consumption over two years by month and day-of-week aggregated across all customers.

Nevertheless, when the aggregated consumer was grouped on a day-of-week and an hour-of-day basis, we observed a higher variation in energy consumption during the day. Figure 4.6 indicates higher peaks occurs during the evenings between 5 p.m and 9 p.m, which reaches around 1000 KWh or more; while it decreases at least by 70 per cent (from 1000 to 300 KWh) after midnight and until 6 a.m.

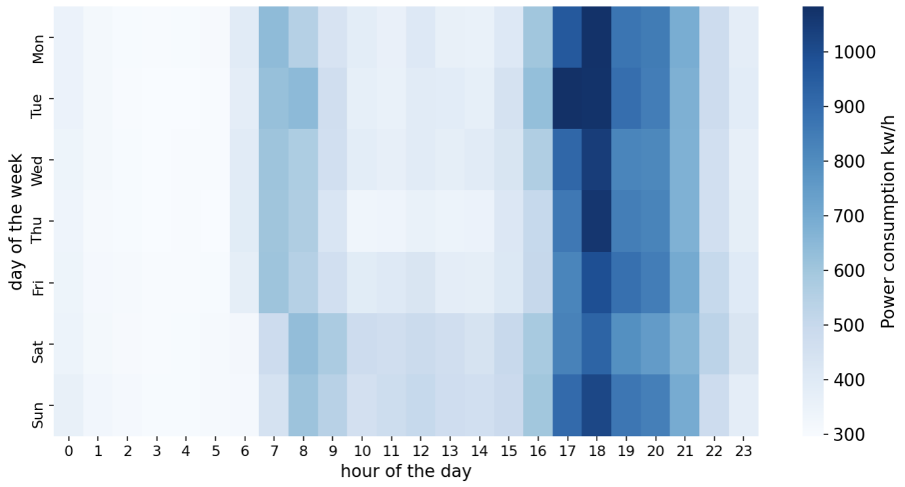


Figure 4.6: Total power consumption over two years by hour-of-day and day-of-week aggregated across all customers.

Feature Extraction

Not all ML algorithms can handle the Date-Time feature and perform well; for this reason, feature extraction is essential to ensure that our ML algorithms perform optimally. The purpose of this step is to extract the data contained in the original features. Besides, through data exploration and analysis, we were able to derive the following features:

- **hour-of-day**: Figure 4.6 demonstrates that the energy consumption varies by hour, and there are ups and downs patterns, which are vital for classifying the anomalies tasks.

- **minute-of-hour**: considering that the dataset is collected every half-hour, this feature is critical to avoid confusion caused by repeating the hour-of-day feature over two half-hour periods. Table 4.2 illustrates an example of features as mentioned above.

Custom-IDs	hour-of-day	minute-of-hour	General Consumption
7	11	00	0.13
7	11	30	0.27

Table 4.2: Example shows hour-of-day and minute-of-hour features.

- **day-of-week**: 4.5 and 4.6 figures display the differences in the energy usage between the days of the week. The day-of-week is an important feature to be included when training a machine learning algorithm.

- **day-of-month**: it holds the information of which day of the month the consumption occur; we included this feature since the clients' activity patterns vary throughout the month.

- **month-of-year**: Figure 4.5 shows that consumers power demand varies from a month to another; this feature is included to enable ML algorithms to capture monthly patterns.

- **year**: It is undoubtedly from figure 4.3 that including the year as a feature is essential since the customers have various habits over the years.

Feature Transformation

After the features extraction step, we ended up with eight features in the dataset. Some of these features are considered categorical since their values are determined by labels, which are *customer-ID* and *year* features. ML algorithms usually require numerical variables as input to achieve better performance. As a result, the dataset's categorical features transformed into numerical features before being used to fit and assess models. The method used for the transformation is called **Ordinal Encoder** from ***Scikit-learn*** module, which converts the categorical variables into ordinal integers. The output will be a column of numbers ranging from 0 to (number of categories - 1); for example, the *year* feature has three values (2011, 2012 and 2013), and it will transform to (0, 1 and 2) respectively. Additionally, most of the techniques used in this project are from ***Scikit-learn*** module, which is a python package that integrates a diverse set of state-of-the-art ML methods for solving supervised and unsupervised problems, aiming to make machine learning accessible to non-specialists through the use of a general-purpose high-level language [45, 46].

4.3.2 Anomaly Identification

As mentioned in the previous chapter, The efficiency of the *Prophet* method to address any seasonality present in time-series data is the primary objective of using it. Power consumption data is closely related to human activities, which are influenced by various factors such as holidays, weather seasons, and day/night rhythm. For instance, individuals may choose to stay at home on weekends and travel on holidays. These historical patterns in power usage may be analysed and utilised to identify outliers. Consequently, we employed the *Prophet* method to analyse and identify the outliers by applying the anomaly decision rules mentioned in chapter 3.

Implementing an algorithm is usually required parameters tuning step, which is the task to select the best collection of parameters for a learning method; these parameters values regulate the learning process and refer to as Hyperparameters. Since we had historical energy consumption data of ten different customers, we tuned the hyperparameters of the Prophet method for each of them separately, as each customer had varying power demands.

First, we installed the required ***fbprophet*** package into the *anaconda* environment to perform the *Prophet* method. Following that, we prepared the dataset for modelling by choosing only the *Date-Time* and *General Consumption (GC)* features and renaming them ***'ds'*** and ***'y'***, respectively, as the *Prophet* demanded. Subsequently, we used a time series cross-validation technique to tune the hyperparameters. The *Prophet* package offers this technique for calculating forecast inaccuracy using historical data, which is accomplished by picking "cutoff" points in the history and fitting the model using the data up to those "cutoff" points. After fitting the model, a prediction is formed for a particular period referred to as the "horizon". The forecasted numbers can then be compared to the actual values to evaluate the current model. The "cutoff" points can be defined as a range of historical points, either automatically or manually [21], considering the following :

- The "initial" training period is chosen to be three times the length of the "horizon" forecasting period, and its size required to be large enough to capture the seasonality being studied. For instance, if the yearly seasonality is included in the model, the initial period must be at least one year in length [21].

- Cutoffs can be made every half a "horizon" [21].

With the range of two years data and taking the mentioned consideration into account, we determined an initial training period as 18 months, a horizon forecasting period as six months and a single cutoff point of "01-12-2012".

The *Prophet* module provides a *performance metrics* tool that can be used to select the hyperparameter combination set with the lowest evaluation measure. This utility analyses each hyperparameters combination forecast and calculates some valuable statistics on the prediction performance, such as the ***root mean squared error (RMSE)***. The *RMSE* is a commonly used metric to assess the difference between the actual and forecasted values; the smallest *RMSE* indicates that the model performs the best.

To tune the hyperparameters, we created a dictionary of all rational hyperparameter combinations. The parameters included in the tuning part is:

- *change_point_range:* This is the percentage of time in history during which a trend is permitted to alter. This parameter prevents overfitting from trending changes towards the time series's end, where there is a limited runway for fitting it correctly. For instance, if the parameter is set to 0.8 (80%), the model will not fit any trend changes occurred during the final 20% of the time series [21].
- *changepoint_prior_scale*: It establishes the trend's flexibility, more precisely how much the trend varies at trend changepoints. This parameter is the most significant. The too-small value of the parameter will lead to trend underfitting, and too large the parameter will lead to a trend overfitting [21].
- *seasonality_prior_scale*: This setting affects the seasonality's adaptability. Similarly, a large value of the parameter permits the seasonality to accommodate substantial swings. In contrast, a small value of the parameter reduces the seasonality's amplitude [21].
- *holiday_prior_scale*: This regulates the degree of flexibility to holiday impacts, as with the *seasonality_prior_scale* [21].
- *seasonality_mode*: There are two options: *additive* or *multiplicative*. By default, *Prophet* accommodates additive seasonalities, which means that the seasonality's influence is added to the trend to get the prediction. The multiplicative seasonality grows with the trend, and it is not a constant additive component [21].

After optimising the hyperparameters for each customer's data and obtaining the optimal settings, three primary functions are run over each of the ten customers' data separately.

The three functions are responsible for fitting and predicting the model, identifying and plotting the anomalies.

- *fit_predict_model*: this function initialised the *Prophet* model and added the necessary seasonality. Including the built-in components such as daily_seasolaity, yearly_seasonlity and add_country_holidays. Besides the manually implemented week_days and week_ends seasonality. We set the interval_width parameter to 0.99 to obtain the 99 per cent of confidence interval, in addition to the tuned hyperparameters. Further, we fitted the model on the whole data of the individual customer, then predicted the entire data to yield the estimated general consumption $\hat{y}$ and the associated $\hat{y}_{upper}$, $\hat{y}_{lower}$ values. Finally, visualised the model's components. The following figures show an example of the *Prophet* components using customer (number seven) data. The actual data is depicted in Figure 4.7 as black dots, whereas the predicted power usage is depicted as a dark blue line. Simultaneously, the light blue boundaries denote the $\hat{y}_{upper}$, and $\hat{y}_{lower}$ values, while the region between them denotes the confidence interval.

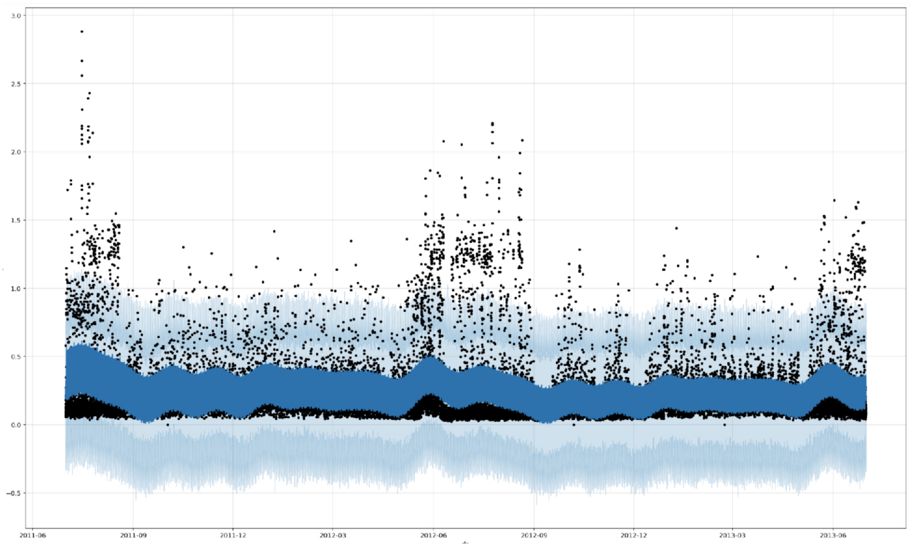


Figure 4.7: The Prophet model's predictions.

The trend changes are visualised in Fig 4.8, starting with a peak in 2011 and decreasing to a low in 2013. The holiday component, seen in Figure 4.9, Figure 4.10 captures the yearly seasonality, while the daily seasonality is displayed in fig 4.11, and the week_days and week_ends components are presented in 4.12.

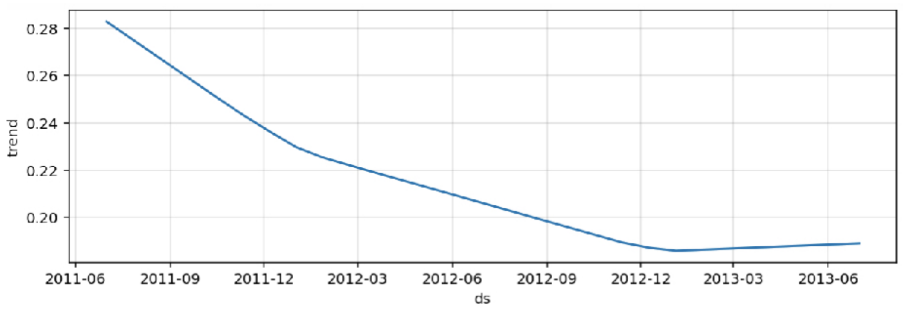


Figure 4.8: The trend of the Prophet model.

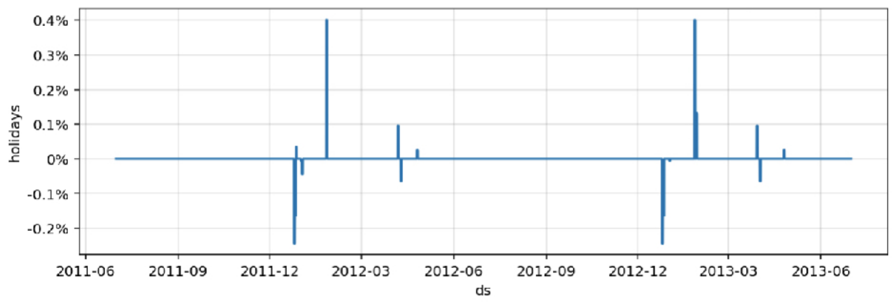


Figure 4.9: The holidays component of the Prophet model.

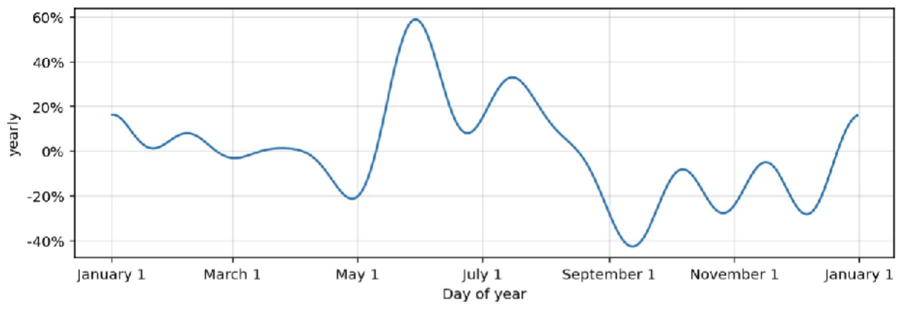


Figure 4.10: The yearly component of the Prophet model.

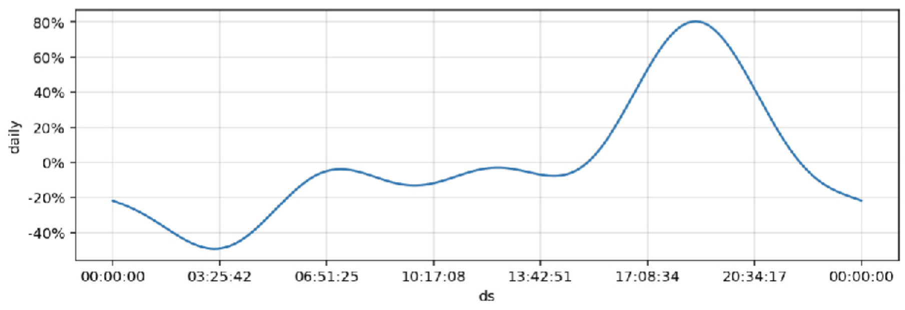


Figure 4.11: The daily component of the Prophet model.

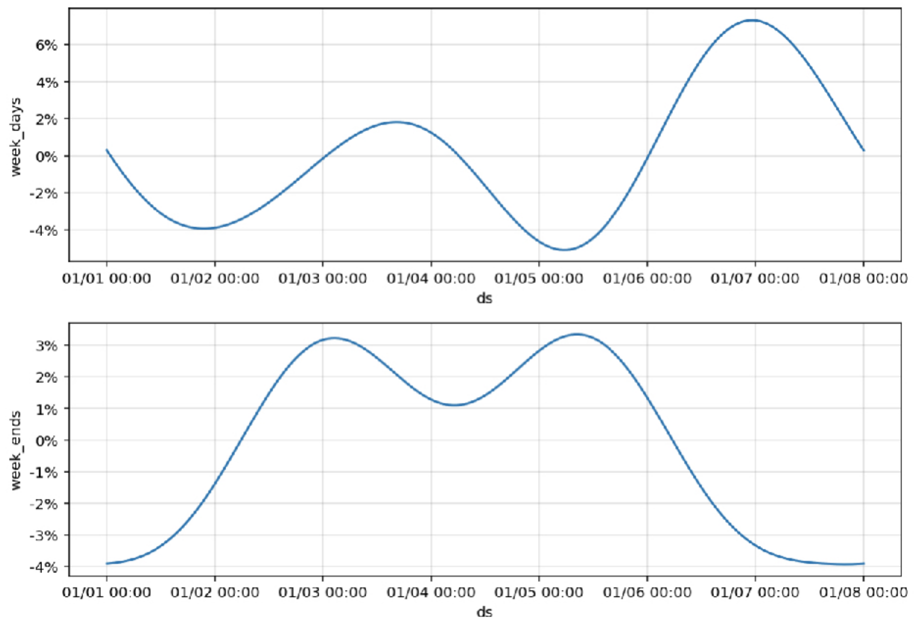


Figure 4.12: The week_days and week_ends components of the Prophet model.

- *detect_anomalies*: in this function, we utilised the forecasting results from the preceding function as an input to identify the anomalies, where any point beyond the uncertainty interval is defined to be anomalous. Following the next decision rules:

  - if the true point $> \hat{y}_{upper}$, anomaly_class $\Rightarrow$ 1.
  - if the true point $< \hat{y}_{lower}$, anomaly_class $\Rightarrow$ 1.
  - if the true point $= 0$, anomaly_class $\Rightarrow$ 1.
  - otherwise, anomaly_class $\Rightarrow$ 0.

  This function returns the anomaly classes.

- *plot_anomalies*: this function visualises the data and the anomalies points. Figure 4.13 shows an example of the same customer data we used in the (*fit_predict_model*) section, customer number seven. The figure presents all 35088 rows of data collected during two years. Actual data are represented by a black circle, whereas green circles represent anomalies with a diameter proportionate to their distance from the interval range $[\hat{y}_{lower}, \hat{y}_{upper}]$. In addition, the yellow interval represents the uncertainty interval. Moreover, the plot may look complicated as a result of the massive amount of data points. As a consequence, we exhibit a subset of the data to illustrate the abnormalities in greater depth. Figure 4.14 depicts actual consumption data over three months; on the contrary, figure 4.15 depicts anomalies over the same period.

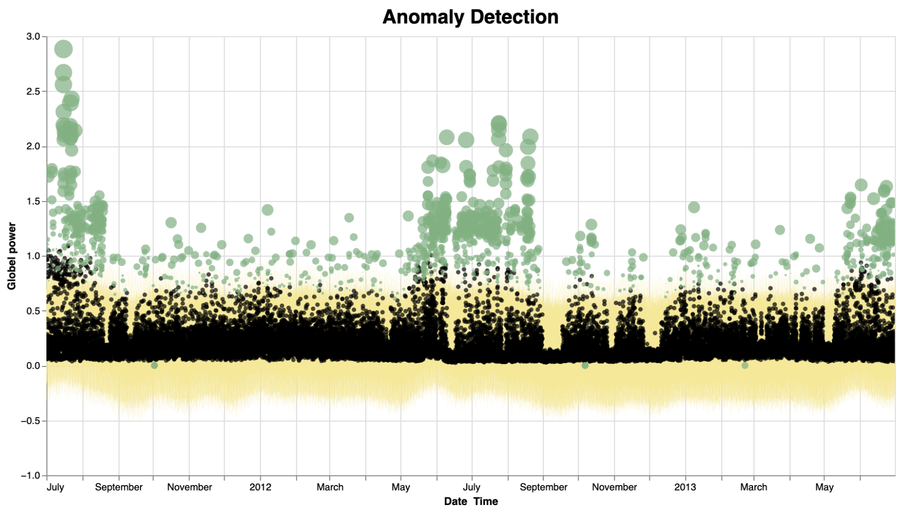


Figure 4.13: The anomalies detected in customer data.

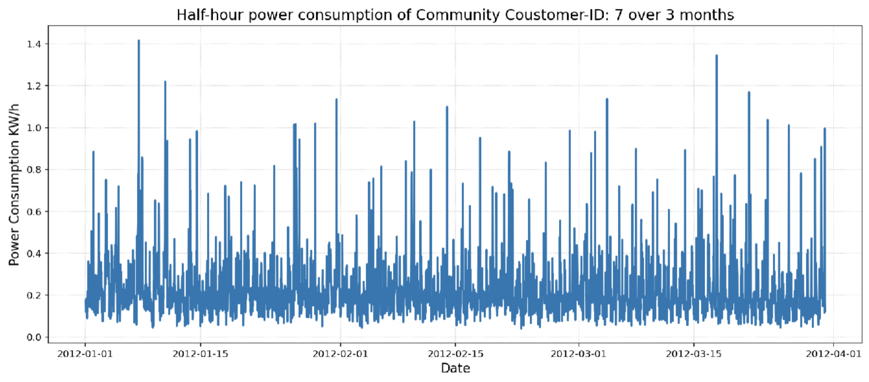


Figure 4.14: Half-hour power consumption of customer over three months.

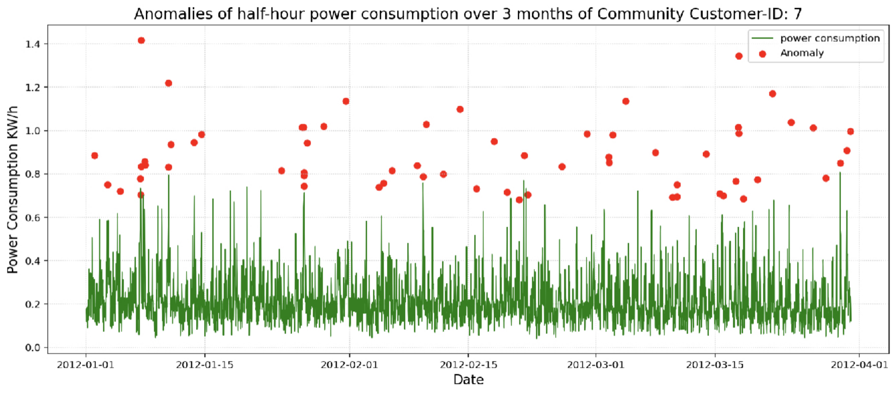


Figure 4.15: Anomalies of half-hour power consumption over three months.

4.4 Phase 2

The first phase's output data is used as input in this step. The dataset contains the nine features mentioned above: customer ID, GC, hour_of_day, minute_of_hour, day_of_week, day_of_month, month_of_year, year, and Anomaly_class. We concatenated together all the data of the ten selected customers to create 350888 rows of labelled power consumption data. However, when we computed the percentage of data with a negative class (Anomaly class = 0), the result was 96.6 %. By contrast, the positive class (Anomaly class = 1) accounts for 3.4 % of the total data, resulting in an imbalanced ratio of 0.036 %, which is referred to as a severe imbalance distribution. Additionally, the minority class has a 3:100 class distribution, which significantly skewed the class distribution. When data exhibits these characteristics (severely imbalanced and skewed distribution), it necessitates the use of particular techniques to correctly manage the minority class, as discussed in Chapter 3. Figure 4.16 depicts how the data is disproportionately distributed across the two classes. Further, the link between the general consumption feature and the Anomaly class feature is illustrated in figure 4.17, which clearly demonstrates how the anomaly class consumes more and more power than the normal class, as expected.

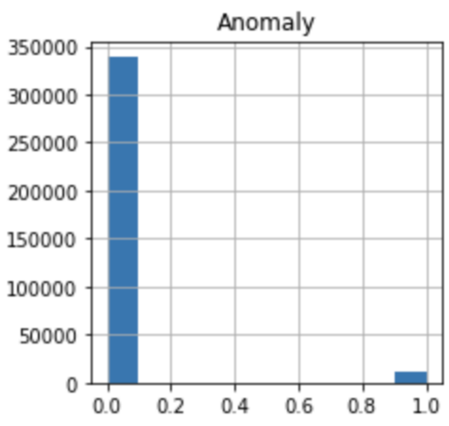


Figure 4.16: The histogram of the data classes.

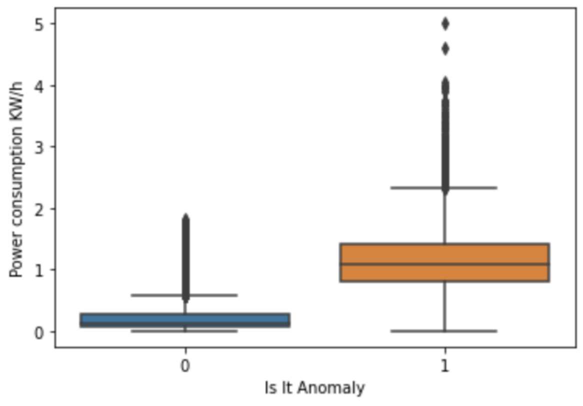


Figure 4.17: The relationship between power consumption and class variable.

This phase discusses the experimental setup, the imbalanced data preparation procedures used in our research and the suggested ML models, and how they were executed and verified. Finally, it discusses the evaluation measures implementation.

4.4.1 Experimental Setup

Model assessment involves more than simply evaluating a model; it also entails evaluating alternative data preparation techniques, alternative learning algorithms, and alternative hyperparameters for optimal learning algorithms [12]. Hence, we split the data as 70/30 for a train set and test set to achieve the testing of all possible alternatives. ***Scikit_learn*** has a helper function (**train_test_split**) to split the data into training and testing sets randomly. This function has a significant feature for imbalanced data, whereby each set can have about the same proportion of samples from each target class as the whole set [45, 46]. The next section of the code is used to split the data into the train and test sets with precise percentages of 96.6% for the negative class and 3.4% for the positive class.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df.loc[:, ~df.columns
    .isin(['Anomaly'])], df.Anomaly, test_size=0.3, random_state=4,
    stratify=df.Anomaly,shuffle=True)
```

Splitting the dataset

Moreover, to experiment with the train set with different data preparation designs, we used the ***Scikit_learn*** function (**RepeatedStratifiedKFold**). This function split the training set into k folds and guarantees that the original distribution's proportion of positive to negative samples is maintained across all folds. A model is trained with the first k - 1 folds, while the remaining fold is used to validate the model. The procedure is continued until the model is validated using each of the folds; then, the model's performance is calculated as the average of all k folds runs [12]. The following code illustrates this procedure.

```python
from sklearn.model_selection import RepeatedStratifiedKFold

# model: is any of the ML models that we would like to test.
# f1 is a model evaluation's measure

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(model, train_set, train_labels_set, scoring='f1'
    , cv=cv)
print('Mean F1_score: %.3f' % mean(scores))
```

The learning process with a train set.

Furthermore, to find the optimal hyperparameters of each model under consideration, we examined two techniques. First, we used **GridSearchCV** from ***Scikit_learn***, which examines all possible parameter combinations comprehensively. The function produces its candidates using a grid of parameter values defined with the *param_grid* parameter. In addition, it evaluates all the possible combinations and returns the best combination. In the code below, we display an example of how this function is implemented [45, 46]. We used the GridSearchCV with the mentioned above K-folds cross-validation method.

```python
from sklearn.model_selection import GridSearchCV

# model: is any of the ML models that we would like to test.
# f1 is a model evaluation's measure

param_grid = dict()
param_grid['model__parameter1'] = ["parameter1's values"]
param_grid['model__parameter2'] = ["parameter2's values"]

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
search = GridSearchCV(model, param_grid ,scoring='f1', cv=cv)
best_model = search.fit(train_set, train_labels_set)
```

Using the GridSearchCV for hyperparameters optimisation.

Secondly, we used Bayesian hyperparameter optimisation. This method chooses its parameter combinations by concentrating on those regions of the parameter space that might yield the most validation scores. This method considers previous assessments when determining which hyperparameter set to evaluate next [47]. Additionally, this method requires four main components; the **Hyperopt** python library is utilised to implement it:

- *The search space of hyperparameters* can be defined as a distribution or a list of categories of choices [47], as illustrated in the following code.

```python
 # import the libraries
 from hyperopt import tpe, STATUS_OK, Trials, hp, fmin

 ''' Domains of parameters can be created using one of many
Hyperopt-specific distribution functions. A hp.choice can be used
 to indicate parameters consists of a list of categories.'''
 space = { 'parameter1': hp.choice('parametert1', ["parameter1's
values"]) ,'parameter2' : hp.choice('parameter2', ["parameter2's
values"]) }

```

The Bayseian search space of hyperparameters example.

- *The objective function* evaluates hyperparameters set to maximise the objective measure [47], which is the "F1" measure score in our case. The next code shows an example of this function.

```python
  def objective(params):
     # Use early stopping and evaluate based on f1-score.
     # model: is any of the ML models that we would like to test.
     # f1 is a model evaluation's measure.

     scores = cross_val_score(model, train_set, train_labels_set,
 cv=5, scoring='f1')
     # Extract the best score
     best_score = max(scores)
     # Loss must be minimised, as maximising the f1 score is
equal to minimising the loss
     loss = 1 - best_score
     return {'loss': loss, 'status': STATUS_OK}

```

The Bayseian objective function example.

- *The surrogate function* suggests parameter choices for the objective function that probably increase the abjective measure's score. In this project, *Tree Parzen Estimator (TPE)* is used as a surrogate function, which is a probabilistic model that maps hyperparameters to the likelihood of achieving a particular score on the objective function [47].

- *The selection function* defines the criterion on the surrogate function to suggest the hyperparameters set to the objective function. The TPE-based surrogate function uses a common metric referred to as *Expected Improvement* [47].

  The following code displays an example for both functions.

```
# Tree Parzen Estimators Algorithm tpe (the surrogate function)
tpe_algorithm = tpe.suggest

# Trials: This method saves the initialization data and the
dictionary returned by the objective function.
model_trials = Trials()

# Setting up the surrogate and selection functions in Hyperopt
MAX_EVALS = 25 # max number of iterations
best_parms = fmin(fn = objective, space = space, algo = tpe.
suggest, max_evals = MAX_EVALS, trials = model_trials)

```

The Bayseian surrogate and selection functions example.

As explained at [47], each iteration follows three main steps, which are:

- Determine the set of hyperparameter values that maximises the Expected Improvement by optimising the selection function over the surrogate function, .
- Submit this hyperparameter combination for evaluation to the objective function, and obtain the appropriate score.
- Update the surrogate function along with the objective function's feedback.

4.4.2 Phase 2 - Data Preprocessing

Phase 2 - preprocessing methods are used to enhance the performance of machine learning models, either through a reduction in execution time or an improvement in the scores of assessment measures. Two primary techniques were employed and examined with our models, Data sampling and Data scaling.

Data Sampling

One of the categories to address imbalanced classification problems is Data-Level approaches which include Data sampling technique; this method modifies the training examples to provide more balanced class distribution, allowing classifiers to perform similarly to conventional classification [48]. In chapter 3, we reviewed the three groups of data sampling. However, the method utilised in this project is based on the suggestions and findings of the [48] publication.

Batista et al. [48] discuss and analyse the behaviour of several data sampling techniques to address imbalanced datasets. They examine various methods that belong to all data sampling categories. The categories include undersampling, oversampling and hybrid sampling (oversampling and undersampling combinations). Furthermore, their findings indicate that hybrid approaches produce very good results for datasets with few positive examples. According to the authors, SOMTE + ENN removes more examples than the other hybrid methods, resulting in a more in-depth data cleaning, whereby ENN is used to exclude instances from both classes, and every example that is misclassified by three of its nearest neighbours is eliminated from the training set. Consequently, we employed SMOTE + ENN as a data sampling approach to re-balance the distribution of the data classes.

The **SOMTEENN** ready-to-use class from ***Scikit-imbalanced-learn*** is utilised in the study. Whereas SMOTE generates new instances of the minority class, while ENN reduces noisy points along with class borders. It is worth mentioning that sampling techniques are used only with the train set, not the test set. Figure 4.18 demonstrates the difference in the distribution of classes before and after data sampling.

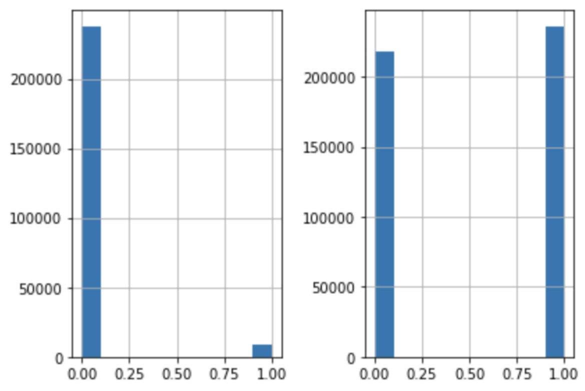


Figure 4.18: The histogram plots of the class distribution for the imbalanced training set and resampled training set.

Data Scaling

Scaling the dataset and approximating the distributions of the individual features to a standard normal distribution is a common criterion of many ML algorithms to improve their performance. As if the data features have various significant variances, the feature of greater variance may dominate a model's objective function and prevent the model from learning from other features appropriately [45, 46].

The ***Scikit-learn*** **StandardScaler** class is examined with some of our models since the features' variances vary significantly. The **StandardScaler** is eliminating the mean and scaling to unit variance by computing the mean and standard deviation on each feature individually; subsequently, the standard score of sample x can be calculated by subtracting the sample from the mean of its feature and dividing the answer on the standard deviation [45, 46]. Bearing in mind that the data scaling must be applied on both train and test sets.

4.4.3 Cost-Sensitive Logistic Regression Model

The ***scikit-learn*** library has an implementation of a logistic regression algorithm called **LogisticRegression**, which support weighted class. The model has a *class_weight* hyperparameter, which can be set to a "***balanced***" value or set to a dictionary with weights associated with dataset classes. The "***balanced***" option automatically adjusts weights inversely proportional to class frequencies in the input data using class labels' values. The "***balanced***" class weights can be calculated as follows [45, 46]:

$$\text{class\_weight of a class} = \frac{\text{Number of total samples}}{\text{Number of classes} \times \text{number of samples belongs to the class}} \tag{4.1}$$

Experimental Design

We examined four alternative weighted logistic regression models with the training set. Three runs of 10-fold stratified cross-validation are adopted to evaluate their performance. The F1-score is computed in each iteration of cross-validation. Subsequently, the four LR models are compared by the model's average F1-score and execution time. The comparison is involved each of the following models:

- Balanced class weighted LR model with original training dataset.
- Balanced class weighted LR model with (SMOTEENN) resampled training dataset.
- Balanced class weighted LR model with a standardised training dataset.
- Balanced class weighted LR model with a standardised and resampled training dataset.

Cost-sensitive Logistic Regression Hyperparameters

The two mentioned methods of hyperparameters optimisation are used to tune the parameters of weighted logistic regression. The reason for utilising both approaches is to allow for a comparison between them. We were particularly interested in tuning the following hyperparameters since it is critical to examine with a variety of different values in order to determine which values improve the performance of the model:

- Class_weight: It is used to indicate the weights assigned to classes. It is possible that different class weighting would result in improved performance [46].
- C: It is the inverse of the strength of regularisation, which must always be a positive float [46].
- Solver: indicates which algorithm should be used to solve the optimisation problem [46].
- Penalty: is used to define the standard that will be utilised in penalising (regularisation) [46].

4.4.4 DBSCAN Model

Clustering is performed on our dataset using the **DBSCAN** model from the ***Scikit-learn*** package. The model requires tuning of two critical parameters, namely *eps* and *min_samples*. Additionally, the default *metric* parameter option is set to "**euclidean**", which uses the Euclidean distance measure to determine the distance between samples.

Experimental Design

We started the experiment by defining the hyperparameters' values based on heuristics. Authors in [49] stated that the *min_samples* parameter's purpose is to smoothen the density estimate. They suggested setting the parameter to twice the dataset's dimensions; for instance, the dimensions of our dataset excluding the class feature are 8; hence the *min_samples* value $= (2.dim = 2.8 = 16)$. While [28] proposed determining *eps* value by using the nearest neighbour (KNN) algorithm. KNN method is a supervised ML algorithm that gathers data points to their nearest cluster utilising a preset number of clusters given by the k parameter. In addition, we assigned the *min_samples* value to the k parameter; after that, we plotted the average distance between each point and its nearest neighbours. The *eps* value is selected at the graph's knee point using **KneeLocater** from the ***kneed*** library. The 4.19 figure shows an example of the average distance plot.

Moreover, we explored two different DBSCAN models: one that utilised the original training dataset and another that used a standardised training dataset. The models

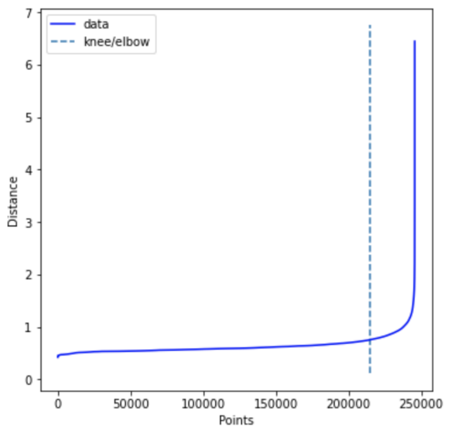


Figure 4.19: The knee and KNN distance plot.

were trained throughout the complete training set and provided clustering labels used to calculate the F1-score. Additionally, the comparison is made using the F1-score and execution time of the models under consideration.

DBSCAN Hyperparameters

We examined both hyperparameters optimisation methods with the DBSCAN model, to compare between them. Further, the two parameters that considered in the tuning experiment are:

- *eps*: It is the maximum distance between two instances where a point can be considered a neighbour of another. Which is not a limit on the maximum distance between points inside a cluster [46].
- *min_samples*: the total number of neighbours for a point to be deemed as a core point, including the point itself [46].

4.4.5 Ensemble Random Forest Model

The **RandomForestClassifier** class from ***Scikit-learn*** is used for implementing the ensemble random forest algorithm.

Experimental Design

With the training dataset, we evaluated seven different ensemble Random Forest models. As with logistic regression trials, their performance is evaluated using three rounds of 10-fold stratified cross-validation. Each cycle of cross-validation is evaluated using the

F1-score. Besides, The number of trees in the forest was defined as 100 for all compared models. Following that, the average F1-score and execution time for the seven RF models are compared. Each of the following models was included:

- Standard Random Forest model with the original training dataset.
- Standard Random Forest model with (SMOTEENN) resampled training dataset
- Balanced class-weighted Random Forest model with the original training dataset.
- Balanced class-weighted Random Forest model with (SMOTEENN) resampled training dataset.
- Bootstrap class-weighted Random Forest model with the original training dataset.
- Bootstrap class-weighted Random Forest model with the (SMOTEENN) resampled training dataset.
- Random Forest model with (Random Undersampling) resampled training dataset.

Furthermore, Jason Brownlee proposed in [12] a modified random forest model called ***Balanced Random Forest***, which uses bootstrap data sampling to alter the class distribution explicitly. As by using this modification we would anticipate a more significant influence on model performance. Hence, we evaluated this model (the last model in the previous list) using **BalancedRandomForestClassifier** class from the ***Scikit-imbalanced-learn*** library; this class randomly undersamples the majority class in each bootstrap sample. Further, we did not investigate the standardisation step because the nature of RF does not require any scaling preprocessing.

Ensemble Random Forest Hyperparameters

The **RandomForestClassifier** class has many hyperparameters, and good results are usually obtained by applying their default values; however, these settings are frequently not optimal and may result in models that consume a significant amount of RAM. Hence, cross-validation should be used to determine the optimal parameter values [46]. The following list summarizes the parameters that are vital to be adjusted.

- *n_estimators*: is the forest's trees number. When it is set to a large number, the results are improved; however, the model takes a longer time for computation. Besides, the outcomes will plateau after a certain number of trees [46].

- *criterion*: The quality of a split is determined by this parameter. The terms "*gini*" and "*entropy*" are used to refer to the *Gini impurity* and the *information gain*, respectively [46].
- *max_depth*: is the tree's maximum depth. If "*None*" is specified, nodes are extended until all leaves are either pure or contain less than *min_samples_split* samples [46].
- *min_samples_split*: is the minimum sample size necessary to divide an internal node [46].
- *min_samples_leaf*: is the bare minimum number of samples that must be present at a leaf node. Any split point, regardless of depth, will only be evaluated when each of the right and left branches has at least *min_samples_leaf* of training samples at leaves [46].
- *max_features*: is the size of random feature subsets to consider when dividing a node. The smaller number leads to a greater reduction of the variance, while a larger number can increase the bias [46].
- *class_weight*: is the weights associated with classes. We consider two options of this parameter values, which are "*balanced*" and "*balanced_subsample*". The former automatically adjusts weights inversely proportionate to class frequencies in the input data based on the values of class labels; it can be calculated as equation 4.1. The latter is identical to "*balanced*", except that weights are generated for each tree built using the bootstrap sample [46].

4.4.6 One-Class Isolation Forest model

Scikit-learn offers an implementation of the isolation forest algorithm called **Isolation-Forest**. This class isolates data points by randomly choosing a feature and a split-value in a range between [minimum value, maximum value] of the selected feature. Further, the number of splitting required to isolate a sample is set to the path's length from the root to the terminal node. Additionally, random partitioning significantly reduces the pathways taken by anomalies, and any specific sample with shorter path lengths is considered to be anomalous [46]

Experimental Design

We divided the training set into *train* and *validation* sets to fit and predict the iForest models. Based on their F1-score and run duration, we investigated and contrasted two models. The first model was trained on the new training set and predicted using the validation set. In contrast, the second model was only trained over the new training set's majority class and predicted the labels of the validation set. As we know the percentage of positive class samples to negative class samples, we set the contamination parameter of iForest models to 0.01. This parameter is used to define the number of the dataset's outliers.

Isolation Forest Hyperparameters

The following hyperparameters are critical for optimizing iForest performance, as specified in [46]:

- *n_estimators*: is the ensemble's number of base estimators.
- *max_samples*: is the number of samples obtained from a train set for training each base estimator.
- *contamination*: is the fraction of data points that contain outliers. This variable is used to determine the threshold for the samples' scores during fitting, and it should be between $[0, 0.5]$.
- *max_features*: is the max number of features extracted from a train set for each base estimator's training.
- *bootstrap*: When set to True, individual trees are fitted using replacement on random subsets of the training data. While if set to False, no replacement sampling is conducted.

4.4.7 Models Evaluation

To evaluate the models, we created a function called (**evaluation**) that computes all the evaluation measures discussed in chapter 3.

We started by using ***Scikit-learn*** **Confusion_Matrix** metric to compute and return the *True Negative TN*, *False Positive FP*, *False Negative FN*, and *True Positive TP* values. After that, the *Sensitivity*, *Specificity*, *G-mean*, *True Positive Rate TPR* and *False Positive Rate FPR* scores can be computed using the Confusion_Matrix returned values, as explained in chapter 3.

Additionally, we computed the *weighted Precision*, *weighted Recall* and *weighted F1* scores by utilising ***Scikit-learn*** **precision_score**, **recall_score** and **f1_score** metrics, respectively. Besides, a weighted score indicates that a score is calculated by the metric for each class and averaged across the support samples (the number of actual samples for each class). Moreover, the *F2 score* is computed by using **fbate_score** from ***Scikit-learn*** metrics. This score is vital as we concentrated on the positive class. These metrics require the true and predicted labels as input.

Furthermore, we used the **roc_curve** and **roc_auc_score** functions from ***Scikit-learn***. The first function enables us to visualise the ROC curve of our models, which are calculating the probabilities of the estimated class labels (all the models except DBSCAN). The second function computes the area under the ROC curve, which can be added to the ROC plots to have an excellent evaluation of the models understudied.

Finally, we used a Core layer machine with a processor of 2 GHz Quad_Core Intel Core i5 and 16 GB RAM to measure the running time of our models.

Chapter 5

Experimental Results And Discussion

5.1 Anomaly Detection

As we pointed out in chapters 3 and 4, the Prophet algorithm is an additive model which combined a collection of components for forecasting. It is noticeable from the example in figure 5.1 that the Prophet's annual component captured the monthly peak power demand for customer-ID #7, comparable to the plot in figure 5.2, which depicts total historical power consumption aggregated by month and day of the week. Likewise, the Prophet's daily component example in figure 5.3 demonstrated the peak hours of the day for customer-ID #7, which corresponds to the plot of an hour of the day and day of the week in figure 5.4; both figures display the same insights that the customer consumed more energy during the evening than after midnight.

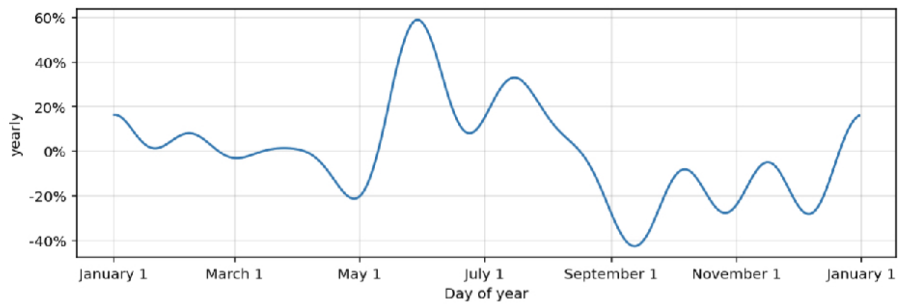


Figure 5.1: The Prophet's yearly component for customer-ID:7.

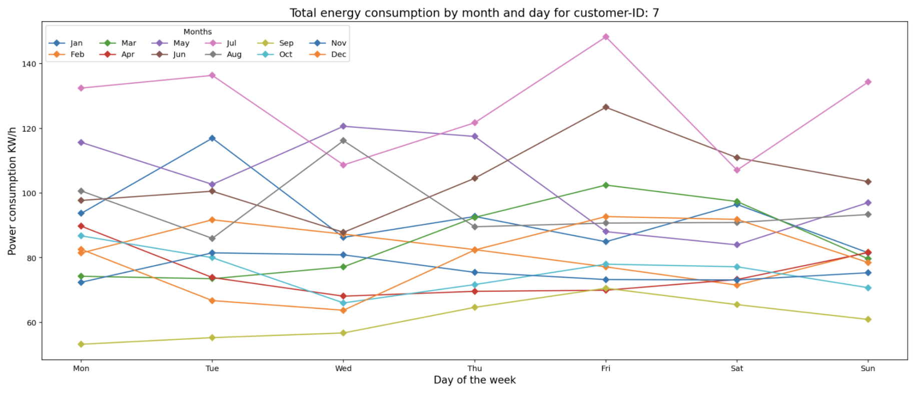


Figure 5.2: The total energy consumption by month and day of the week for customer-ID:7.

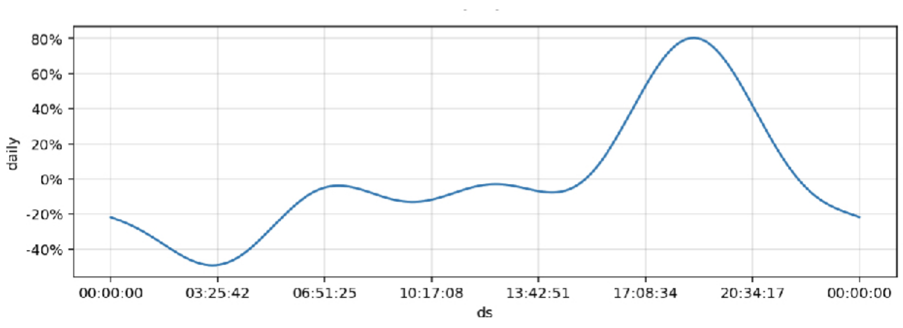


Figure 5.3: The Prophet's daily component for customer-ID:7

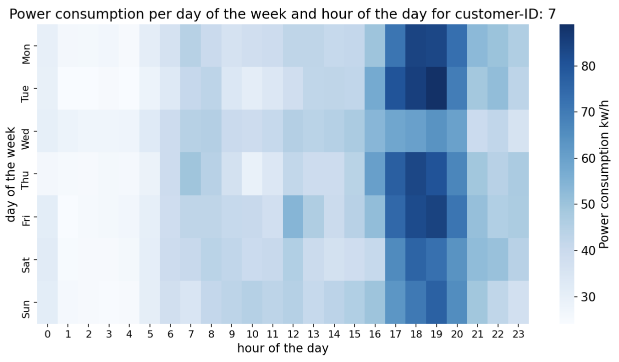


Figure 5.4: The total energy consumption by an hour of the day, and day of the week for customer-ID:7.

The Prophet's model considers all the information and patterns when creating the forecasted data and the uncertainty interval. We chose a 99% uncertainty interval to account for any unexpected events requiring additional energy, such as parties. As seen in figure 5.5, the orange region represents the uncertainty interval, and the blue line denotes the power consumption for customer-ID #7 during a random ten-day period. The red dots indicate locations where the blue line exceeded the predicted range region. These red points describe the outliers in which consumption exceeded expectations.

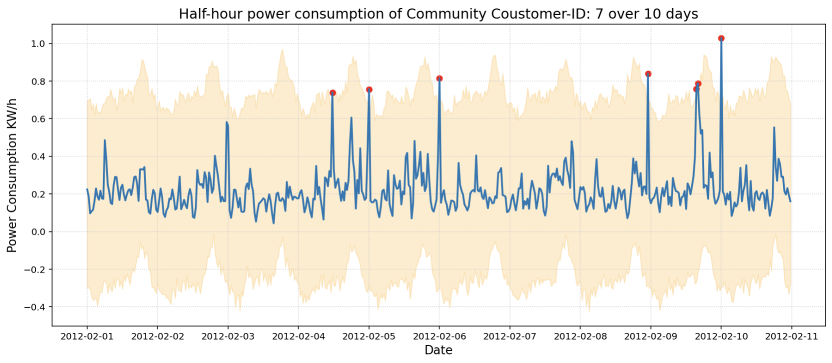
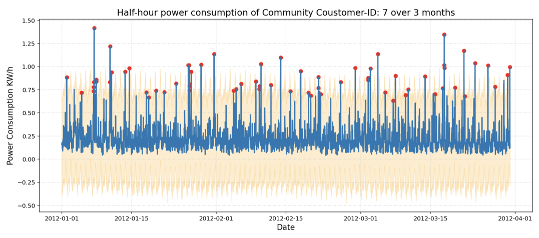
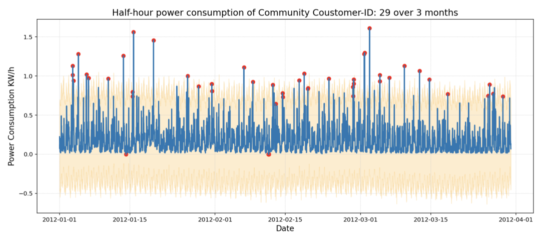


Figure 5.5: A random ten days of power consumption for customer-ID:7, and the Prophet's uncertainty interval.

Furthermore, figure 5.6 displays two graphs of power consumption for two distinct customers; these graphs present the power consumption over three months and the uncertainty interval of the Prophet model, highlighting the locations of the anomalies with red circles. The two customers exhibit various patterns and, therefore, different outliers. Further, graph (B), which belongs to customer #29, contains red circles within the uncertainty interval due to the dataset's zero values for general consumption. Additionally, figure 5.7 illustrates two different customers' energy usage over two years; the black circles indicate actual data, whereas green circles denote anomalies with a diameter proportional to their distance from the interval range. As a result of a great number of data points, the plot may appear compact and unclear.

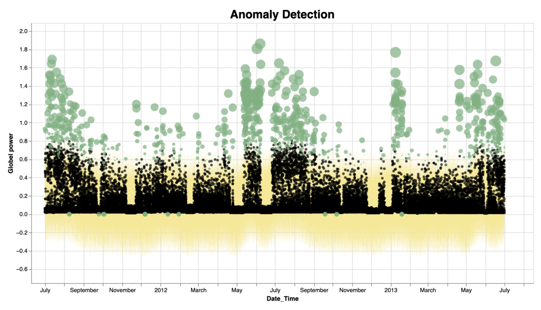


(a) Anomalies of half-hour power consumption over three months of Customer-ID: 7

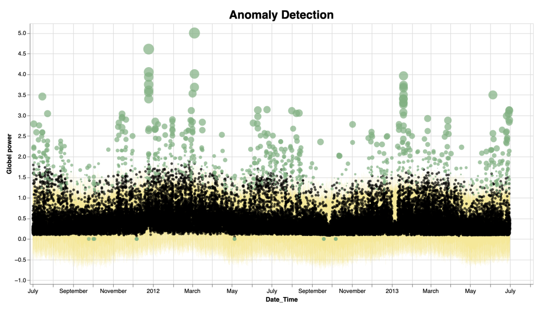


(b) Anomalies of half-hour power consumption over three months of Customer-ID: 29

Figure 5.6: The anomalies of two different clients over three months.



(a) Anomalies of half-hour power consumption over two years of Customer-ID: 30



(b) Anomalies of half-hour power consumption over two years of Customer-ID: 160

Figure 5.7: The anomalies of two distinct customers over two years.

5.2 Imbalanced Classification Experimental Results

This section discusses the experiments conducted on the imbalanced training dataset using the four stated algorithms and various alternative setups. Additionally, this part addresses the evaluation of the best models and establishes the final model for categorising future data points based on the outcomes of the imbalanced classification training.

5.2.1 Grid Search and Bayesian Hyperparameters Optimisations

We evaluated and compared Grid search and Bayesian optimisation as one of the experiments conducted throughout our research. Both approaches were applied to two models: logistic regression and DBSCAN.

Grid search trains and validates the model for each parameter combination supplied, even if the combination area produces unsatisfactory results. Additionally, grid search may be successful when parameter space is limited; otherwise, it takes a considerably longer time. On the other hand, Bayesian optimisation selects a parameter combination based on prior assessments and discards parameter areas that provide undesirable results, which requires a shorter time than grid search. Simultaneously, Bayesian optimisation may avoid attempting a new combination region that may produce better results since it is focused on the area that produces good outcomes [47].

Model Results	Grid Search	Bayesian Optimisation
LR Run-Time	3678.4s	**1486.8s**
LR F2-score	0.667	0.667
DBSCAN Run-Time	20888.06s	**2108.68s**
DBSCAN F2-score	0.155	0.155

Table 5.1: The comparison of the tuning methods.

Table 5.1 compares the execution times of both techniques using LR and DBSCAN models; the findings indicate that Bayesian optimisation spent less time than Grid search while maintaining the same F2-score.

5.2.2 Cost-sensitive Logistic Regression Experimental Results

We compared four models in cost-sensitive logistic regression trials. We began by performing logistic regression with weighted penalty functions on the original training set. In the second model, we used the SMOTEENN sampling approach and the weighted LR on the training set. While, in the third model, we used a standardised training set to execute the weighted LR model. Finally, we used a sampled and standardised training set to run the weighted LR.

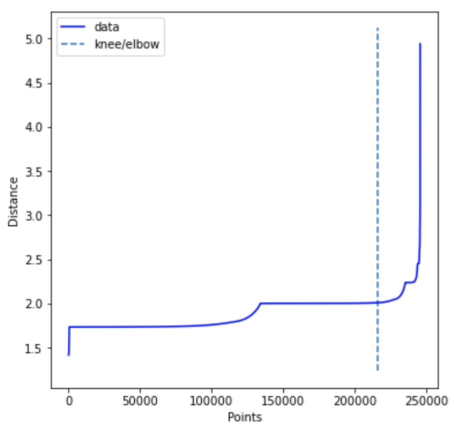
SMOTEENN	Scaling	F1-score	Run-Time
$\times$	$\times$	0.516	28.44s
$\surd$	$\times$	0.958	46.07s
$\times$	$\surd$	0.516	7.39s
$\surd$	$\surd$	**0.958**	**13.15s**

Table 5.2: The Cost-sensitive LR Experimental Results.

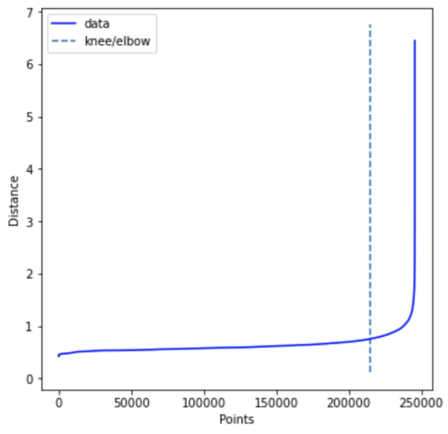
As shown in Table 5.2, the final LR model achieved the highest score based on both the F1-score and the duration time. Additionally, we see that data scaling has no effect on the F1-score but does influence the model's execution time. Furthermore, cost-sensitive LR is a subset of the algorithmic changes for class imbalance discussed in Chapter 3. However, integrating it with a data-level approach such as data sampling produced far better outcomes.

5.2.3 DBSCAN Experimental Results

The reason for using DBSCAN is its capability for detecting outliers. We examined DBSCAN with both original and standardised training data. Further, we avoided data sampling because increasing the number of samples from the minority class will degrade the algorithm's performance and make it incapable of recognising outliers. The following figure shows that scaling the data can affect the KNN distance plot, resulting in various *eps* parameter values. Besides, Scaling the data ensures that all features contribute equally to the model and avoid bias towards features with a larger scale.



(a) DBSCAN with original data



(b) DBSCAN with standardised data

Figure 5.8: The knee and KNN distance plots.

Additionally, Table 5.3 explains how scaling improves the performance of F1-score and duration time, as well as how the *eps* value changes. As the *eps* parameter decreased and the *min_sample* parameter was set to be 16, we observed a rise in the frequency of anomalies and an enhancement in the F1-score.

Scaling	F1-score	Run-Time	eps	min_samples
×	0.075	82.75s	2.008	16
√	**0.575**	**56.97s**	0.752	16

Table 5.3: The DBSCAN Experimental Results.

Furthermore, we tested the algorithm's performance utilising adjusted hyperparameters and heuristics parameters. As seen in table 5.4, the latter consistently outperforms the tuned ones. While the heuristics parameter considered all points, the tuned parameters resulted from several assessments using folds of the training data.

Hyperparameters	F2-score	G-mean
Tuned parameters	0.155	0.358
Heuristics parameters	**0.587**	**0.908**

Table 5.4: The DBSCAN hyperparameters Results.

5.2.4 Ensemble Random Forest Experimental Results

Ensemble random forest merges many classifiers into one classifier without requiring altering the dataset's structure. Considering the nature of the RF method, we did not scale the training data in our trials. The results in Table 5.5 demonstrate that the standard RF has a slightly better F1-score than the balanced class-weighted RF but with a longer duration time. Further, bootstrap class-weighted RF produces a higher F1-score than the latter two. Additionally, the findings reveal that integrating ensemble RF with the SMOTEENN technique is significantly raised the F1-score considerably with or without class-weighting; nevertheless, the execution time is increased. Given that F1-score of balanced and bootstrap class-weighted models were similar, we opted to tune this parameter, with the outcome favouring bootstrap. Finally, the F1-score indicates that standard RF with SMOTEENN is superior to random undersampling, despite the latter's shorter run-time.

Class-weight	Data-sampling	F1-score	Run-Time
×	×	0.857	419.82s
×	SMOTEENN	0.993	1061.18s
balanced	×	0.851	309.03s
balanced	**SMOTEENN**	**0.994**	**910.32s**
bootstrap	×	0.865	444.48s
bootstrap	**SMOTEENN**	**0.994**	1165.81s
×	Random Undersampling	0.606	199.07s

Table 5.5: The Ensemble RF Experimental Results.

5.2.5 Isolation Forest Experimental Results

As referred to in chapter 3, One-class classification iForest belongs to algorithm-level solutions for imbalanced classification tasks, which trains a classifier on the majority class and regards the minority class as outliers. This type of solution does not require any data preprocessing to manage the class distribution skew; as a result, we did not apply data sampling or data scaling. The trials we conducted using iForest are summarized in table 5.6. We began by training the model using all of the training data. In comparison, in the second trial, we trained the model only by using data points from the majority class and verified it using a dataset from both classes. According to the results, the second model has a higher F1-score and runs slightly longer.

Dataset	F1-score	Run-Time
Original dataset	0.296	6.825s
Majority-class dataset	**0.526**	**6.95s**

Table 5.6: The iForest Experimental Results.

5.2.6 The Final Evaluation Results for All Models

Our four models were chosen in accordance with the categories of solution techniques outlined in Chapter 3 for problems involving imbalanced classification. The following stages guided the research for this project:

- Experiment with the four specified algorithms using a variety of different settings on the training dataset.
- Tune the hyperparameters of the best models based on the results of the experiments conducted on the first step.
- Evaluate the best models with the most suitable hyperparameters' values on the test dataset to determine the final model for classifying future data points. The final comparison involved the following models:
    1. Class-weighted logistic regression model, which trained on the standardised and resampled training dataset.
    2. DBSCAN model with the heuristics parameters, which trained on the standardised training dataset.
    3. Bootstrap class-weighted ensemble random forest, which trained on the resampled training dataset.
    4. One-class iForest model, which trained on the majority class training dataset.

According to Table 5.7, ensemble RF has the most significant score in terms of Sensitivity, Specificity, and G-mean, indicating the RF's ability to identify both the negative and, more importantly, the positive classes. Further, the DBSCAN and iForest produce comparable results, whereas cost-sensitive LR produces slightly greater results than the latter two. Furthermore, figure 5.9 demonstrates that ensemble RF outperforms the other models and has the highest F2-score; LR surpasses DBSCAN and iForest, which have similar F2-scores.

Models	Sensitivity	Specificity	G-Mean
Cost-sensitive LR	0.911	0.932	0.921
DBSCAN	0.921	0.896	0.908
Ensemble RF	**0.972**	**0.975**	**0.973**
One-class iForest	0.926	0.895	0.91

Table 5.7: The models' comparisons part-1.

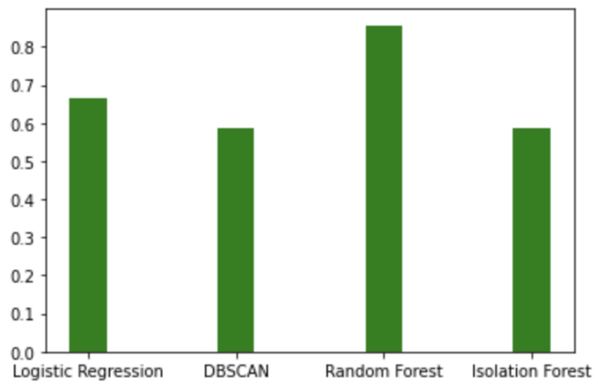


Figure 5.9: The models' F2-scores.

Additionally, we evaluated our models using the weighted Recall, Precision and F1-score. The weighted score generates the score for each class individually, but when combined, they are multiplied by a weight based on the number of actual labels for each class. These measures are focusing on the majority class. Table 5.8 illustrates that RF has the highest score amongst the three measures. On the other hand, figure 5.10 shows the true positive rate TPR and false positive rate FPR for each of the four models. The capability of RF to detect true positives (TPR) is the highest, while its potential to misclassify negatives (FPR) is the lowest of the four models. Further, it is noticeable that DBSCAN and iForest performed similarly and had a superior TPR score than LR; yet, the LR model has a better FPR.

Models	Recall	Precision	F1-score
Cost-sensitive LR	0.931	0.973	0.946
DBSCAN	0.897	0.971	0.924
Ensemble RF	**0.975**	**0.985**	**0.978**
One-class iForest	0.896	0.971	0.924

Table 5.8: The models' comparisons part-2.

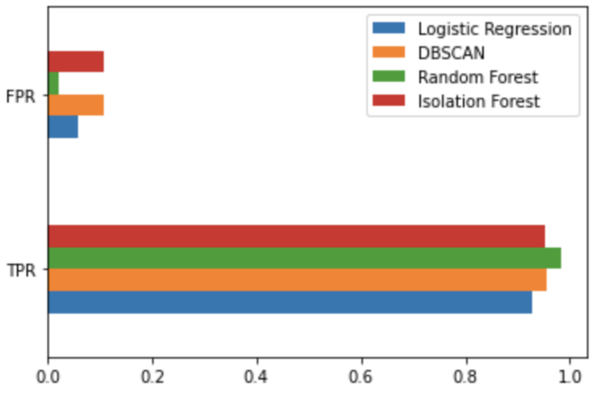


Figure 5.10: The models' True positive rate/ False positive rate.

Moreover, the receiver operating characteristic (ROC) curve is utilized to evaluate the models (except DBSCAN). These models supplied the necessary false positive, true positive rates and threshold values for plotting the curve, unlike DBSCAN. The ROC curves and AUC values for the random forest, iForest, and logistic regression models are shown in figure 5.11. We observe that RF produces highly significant findings from the figure, whereas logistic regression produces the lowest score. On the contrary, from figure 5.12 we notice that logistic regression consumed the shortest execution time, while DBSCAN took the longest time. At the same time, the RF spent a fairly long time than LR, which should be considered when we utilise the model.

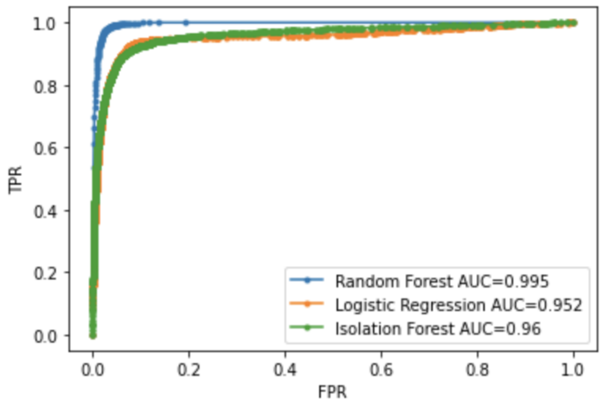


Figure 5.11: The ROC curves and AUC scores.

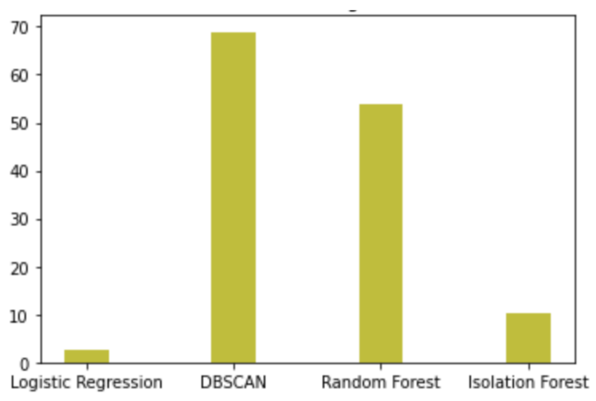


Figure 5.12: The models' execution-times.

As seen by the results and graphs, the ensemble random forest model outperforms all other models. Combining many classifiers as a single classifier in ensemble random forest adds more strength to the imbalanced classification than a single classifier does, as viewed in table 5.5, where the standard random forest gave good results. Additionally, by integrating the random forest model with a class-weighted method to minimise the total cost error for the dataset classes and by preprocessing the data with SMOTEENN, the dataset's class distribution is rebalanced, resulting in a significant improvement in the classification task.

5.3 Future Work

We aim to employ a scheduling cluster system in the future to enhance the speed and reliability of a single computer. We intend to use the entire Ausgrid dataset rather than just ten clients. Additionally, we plan to utilize and analyze the Ausgrid dataset's other categories, such as the controlled load CL category dataset. Moreover, we intend to test our approach using a variety of different datasets in order to detect negative energy usage. We plan to compare our anomaly detection model results with other baseline models like *Twitter anomaly detection* and *Multiuser anomaly detection* approaches.

Chapter 6

Conclusions

This thesis proposed an approach for anomaly detection of smart meters and classifying future smart meter events based on historical power consumption data. This approach is more effective and efficient than the related works mentioned in chapter 2 since it can detect and classify a broader range of anomalies.

We applied an unsupervised learning and statistical algorithm for finding the anomalies based on the customer's power consumption patterns (FB Propthet). This algorithm considered the multi-period seasonality and the influence of external factors on real-world power demand data. Furthermore, after labelling the data with anomalies, we generated a new dataset which is available for further experimentation by anyone. We also experimented with classification approaches on the new dataset with labelled anomalies to find future anomalies. The classification approach achieved a good G-mean score of 97 per cent. For classification, we utilised an ensemble-based machine-learning algorithm for imbalanced classification. The performance and results of the suggested approach have verified its efficacy as a powerful method for detecting smart meter anomalies that could potentially prevent Grid imbalance.

List of Figures

List of Tables

Appendix A

Source Code

The source code of the thesis is available on: https://github.com/Fadwa-Maatug/Master-Thesis-Anomaly-Detection-of-Smart-meter-Data.

There are two sub-folder: **Part-1** and **Part-2**.

- The ***Part-1*** folder contains a Jupyter notebook file with the python code of the first phase of the proposed solution approach (Anomaly detection and labelling the datasets), besides the dataset files and a file of the best hyperparameter values.
- The ***Part-2*** folder contains a Jupyter notebook file with the python code of the second phase of the suggested approach (Imbalanced classification models) and the labelled dataset resulting from the first phase.

Bibliography

[1] Xiaohui Wang, Ting Zhao, He Liu, and Rong He. Power Consumption Predicting and Anomaly Detection Based on Long Short-Term Memory Neural Network. In *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 487–491, April 2019. doi: 10.1109/ICCCBDA.2019.8725704.

[2] Yimin Zhou, Yanfeng Chen, Guoqing Xu, Qi Zhang, and Ludovic Krundel. Home energy management with PSO in smart grid. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pages 1666–1670, June 2014. doi: 10.1109/ISIE.2014.6864865. ISSN: 2163-5145.

[3] Leping Zhang, Lu Wan, Yong Xiao, Shuangquan Li, and Chengpeng Zhu. Anomaly Detection method of Smart Meters data based on GMM-LDA clustering feature Learning and PSO Support Vector Machine. In *2019 IEEE Sustainable Power and Energy Conference (iSPEC)*, pages 2407–2412, November 2019. doi: 10.1109/iSPEC48194.2019.8974989.

[4] Rituka Jaiswal, Antorweep Chakravorty, and Chunming Rong. Distributed Fog Computing Architecture for Real-Time Anomaly Detection in Smart Meter Data. In *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 1–8, Oxford, United Kingdom, August 2020. IEEE. ISBN 978-1-72817-022-0. doi: 10.1109/BigDataService49289.2020.00009. URL https://ieeexplore.ieee.org/document/9179551/.

[5] Xiufeng Liu and Per Sieverts Nielsen. Regression-based Online Anomaly Detection for Smart Grid Data. *arXiv:1606.05781 [cs]*, June 2016. URL http://arxiv.org/abs/1606.05781. arXiv: 1606.05781.

[6] Wen-Xiang Fang, Po-Chao Lan, Wan-Rung Lin, Hsiao-Chen Chang, Hai-Yen Chang, and Yi-Hsien Wang. Combine Facebook Prophet and LSTM with BPNN Forecasting financial markets: the Morgan Taiwan Index. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 1–2, December 2019. doi: 10.1109/ISPACS48206.2019.8986377. ISSN: 2642-3529.

[7] Froogh Fathnia, Farid Fathnia, and D. B. Mohammad Hossein Javidi. Detection of anomalies in smart meter data: A density-based approach. In *2017 Smart Grid Conference (SGC)*, pages 1–6, December 2017. doi: 10.1109/SGC.2017.8308852. ISSN: 2572-6927.

[8] Vikramaditya Jakkula and Diane Cook. Outlier Detection in Smart Environment Structured Power Datasets. In *2010 Sixth International Conference on Intelligent Environments*, pages 29–33, July 2010. doi: 10.1109/IE.2010.13.

[9] Halldor Janetzko, Florian Stoffel, Sebastian Mittelstädt, and Daniel Keim. Anomaly Detection for Visual Analytics of Power Consumption Data. *Computers & Graphics*, 38:27–37, February 2014. doi: 10.1016/j.cag.2013.10.006.

[10] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, February 2019. ISBN 978-1-108-42209-3. Google-Books-ID: CYaEDwAAQBAJ.

[11] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C. Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning From Imbalanced Data Sets*. Springer, Cham, Switzerland, 2018. ISBN 978-3-319-98073-7. URL http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1920612&scope=site.

[12] Jason Brownlee. *Imbalanced Classification with Python*. Jason Brownlee, v1.3 edition, 2021. URL https://machinelearningmastery.com/imbalanced-classification-with-python/.

[13] Oded Maimon and Lior Rokach, editors. *Data Mining and Knowledge Discovery Handbook*. Springer US, Boston, MA, 2010. ISBN 978-0-387-09822-7 978-0-387-09823-4. doi: 10.1007/978-0-387-09823-4. URL http://link.springer.com/10.1007/978-0-387-09823-4.

[14] Wen-Xiang Fang, Po-Chao Lan, Wan-Rung Lin, Hsiao-Chen Chang, Hai-Yen Chang, and Yi-Hsien Wang. Combine Facebook Prophet and LSTM with BPNN Forecasting financial markets: the Morgan Taiwan Index. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 1–2, December 2019. doi: 10.1109/ISPACS48206.2019.8986377. ISSN: 2642-3529.

[15] Anusha Garlapati, Doredla Radha Krishna, Kavya Garlapati, Nandigama mani Srikara Yaswanth, Udayagiri Rahul, and Gayathri Narayanan. Stock Price Prediction Using Facebook Prophet and Arima Models. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–7, April 2021. doi: 10.1109/I2CT51068.2021.9418057.

[16] Bineet Kumar Jha and Shilpa Pande. Time Series Forecasting Model for Supermarket Sales using FB-Prophet. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 547–554, April 2021. doi: 10.1109/ICCMC51019.2021.9418033.

[17] Md. Mehedi Hasan Shawon, Sumaiya Akter, Md. Kamrul Islam, Sabbir Ahmed, and Md. Mosaddequr Rahman. Forecasting PV Panel Output Using Prophet Time Series Machine Learning Model. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 1141–1144, November 2020. doi: 10.1109/TENCON50793.2020.9293751. ISSN: 2159-3450.

[18] Cynthia Freeman, Jonathan Merriman, Ian Beaver, and Abdullah Mueen. Experimental Comparison of Online Anomaly Detection Algorithms. In *Experimental Comparison of Online Anomaly Detection Algorithms*, Sarasota, Florida, USA, May 2019. Artificial Intelligence Research Society Conference.

[19] Karthick Thiyagarajan, Sarath Kodagoda, Nalika Ulapane, and Mukesh Prasad. A Temporal Forecasting Driven Approach Using Facebook's Prophet Method for Anomaly Detection in Sewer Air Temperature Sensor System. In *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 25–30, November 2020. doi: 10.1109/ICIEA48937.2020.9248142. ISSN: 2158-2297.

[20] Ritchie Vink. Build Facebook's Prophet in PyMC3; Bayesian time series analyis with Generalized Additive Models - Ritchie Vink, October 2018. URL `https://www.ritchievink.com/blog/2018/10/09/build-facebooks-prophet-in-pymc3-bayesian-time-series-analyis-with-generalized-a`

[21] Open Source Facebook. Quick Start, March 2021. URL `http://facebook.github.io/prophet/docs/quick_start.html`.

[22] Shravan Vasishth, Bruno Nicenboim, and Daniel Schad. *An Introduction to Bayesian Data Analysis for Cognitive Science*. BOOKDOWN, June 2021. URL `https://vasishth.github.io/Bayes_CogSci/`.

[23] Jeremy Oakley. *Chapter 5 Interval estimates and confidence intervals | MAS113 Part 2: Data Science*. BOOKDOWN, February 2021. URL `http://www.jeremy-oakley.staff.shef.ac.uk/mas113/notes/interval-estimates-and-confidence-intervals.html`.

[24] Anastasios Bellas, Charles Bouveyron, Marie Cottrell, and Jerome Lacaille. Anomaly Detection Based on Confidence Intervals Using SOM with an Application to Health Monitoring. *arXiv:1508.04154 [stat]*, 295:145–155, 2014. doi:

10.1007/978-3-319-07695-9_14. URL http://arxiv.org/abs/1508.04154. arXiv: 1508.04154.

[25] Feng SHEN, Run WANG, and Yu SHEN. A Cost-Sensitive Logistic Regression Credit Scoring Model Based on Multi-Objective Optimization Approach. *Technological & Economic Development of Economy*, 26(2):405–429, March 2020. ISSN 20294913. doi: 10.3846/tede.2019.11337. URL http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=142286599&scope=site. Publisher: Vilnius Gediminas Technical University.

[26] Lili Zhang, Trent Geisler, Herman Ray, and Ying Xie. Improving logistic regression on the imbalanced data by a novel penalized log-likelihood function. *Journal of Applied Statistics*, 0(0):1–21, June 2021. ISSN 0266-4763. doi: 10.1080/02664763.2021.1939662. URL https://doi.org/10.1080/02664763.2021.1939662. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/02664763.2021.1939662.

[27] Lili Zhang, Herman Ray, Jennifer Priestley, and Soon Tan. A descriptive study of variable discretization and cost-sensitive logistic regression on imbalanced credit data. *Journal of Applied Statistics*, 47(3):568–581, February 2020. ISSN 0266-4763. doi: 10.1080/02664763.2019.1643829. URL https://doi.org/10.1080/02664763.2019.1643829. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/02664763.2019.1643829.

[28] S. Wibisono, M. T. Anwar, A. Supriyanto, and I. H. A. Amin. Multivariate weather anomaly detection using DBSCAN clustering algorithm. *Journal of Physics: Conference Series*, 1869(1):012077, April 2021. ISSN 1742-6596. doi: 10.1088/1742-6596/1869/1/012077. URL https://doi.org/10.1088/1742-6596/1869/1/012077. Publisher: IOP Publishing.

[29] Mete Celik, Filiz Dadaser-Celik, and Ahmet Dokuz. *Anomaly Detection in Temperature Data Using DBSCAN Algorithm*. Springer Science+Business, June 2011. doi: 10.1109/INISTA.2011.5946052. Journal Abbreviation: INISTA 2011 - 2011 International Symposium on INnovations in Intelligent SysTems and Applications Publication Title: INISTA 2011 - 2011 International Symposium on INnovations in Intelligent SysTems and Applications.

[30] Hossein Saeedi Emadi and Sayyed Majid Mazinani. A Novel Anomaly Detection Algorithm Using DBSCAN and SVM in Wireless Sensor Networks. *Wireless Personal Communications*, 98(2):2025–2035, January 2018. ISSN 0929-6212, 1572-834X. doi: 10.1007/s11277-017-4961-1. URL http://link.springer.com/10.1007/s11277-017-4961-1.

[31] Yashaswini Hegde and S.K. Padma. Sentiment Analysis Using Random Forest Ensemble for Mobile Product Reviews in Kannada. In *2017 IEEE 7th International Advance Computing Conference (IACC)*, pages 777–782, IEEE 7th International Advance Computing Conference, January 2017. doi: 10.1109/IACC.2017.0160. ISSN: 2473-3571.

[32] Akin Ozçift. Random forests ensemble classifier trained with data resampling strategy to improve cardiac arrhythmia diagnosis. *Computers in biology and medicine*, 41(5): 265–271, 2011. doi: http://dx.doi.org.ezproxy.uis.no/10.1016/j.compbiomed.2011.03.001. URL http://www.proquest.com/docview/863432648?pq-origsite=primo. Num Pages: 7.

[33] Shaghayegh Miraki, Sasan Hedayati Zanganeh, Kamran Chapi, Vijay P. Singh, Ataollah Shirzadi, Himan Shahabi, and Binh Thai Pham. Mapping Groundwater Potential Using a Novel Hybrid Intelligence Approach. *Water Resources Management*, 33(1): 281–302, January 2019. ISSN 0920-4741, 1573-1650. doi: 10.1007/s11269-018-2102-6. URL http://link.springer.com/10.1007/s11269-018-2102-6.

[34] Yoga Pristyanto, Anggit Ferdita Nugraha, Irfan Pratama, and Akhmad Dahlan. Ensemble Model Approach For Imbalanced Class Handling on Dataset. In *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, pages 17–21. IEEE, November 2020. doi: 10.1109/ICOIACT50329.2020.9331984.

[35] Ehdieh Khaledian, Shikhar Pandey, Pratim Kundu, and Anurag K. Srivastava. Real-Time Synchrophasor Data Anomaly Detection and Classification Using Isolation Forest, KMeans, and LoOP. *IEEE Transactions on Smart Grid*, 12(3):2378–2388, May 2021. ISSN 1949-3061. doi: 10.1109/TSG.2020.3046602. Conference Name: IEEE Transactions on Smart Grid.

[36] Kai Song, Yujie Zhou, Hongming Liu, and Nianhao Zhu. Isolated forest in keystroke dynamics-based authentication: Only normal instances available for training. In *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, pages 63–67. IEEE, September 2017. doi: 10.1109/CIAPP.2017.8167061.

[37] Yu Qin and YuanSheng Lou. Hydrological Time Series Anomaly Pattern Detection based on Isolation Forest. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1706–1710. IEEE, March 2019. doi: 10.1109/ITNEC.2019.8729405.

[38] Shaoqing Liu, Zhenshan Ji, and Yong Wang. Improving Anomaly Detection Fusion Method of Rotating Machinery Based on ANN and Isolation Forest. In *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, pages 581–584. IEEE, July 2020. doi: 10.1109/CVIDL51233.2020.00-23.

[39] Alexander I. Filippov, Artem V. Iuzbashev, and Alexey S. Kurnev. User authentication via touch pattern recognition based on isolation forest. In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 1485–1489. IEEE, January 2018. doi: 10.1109/EIConRus.2018.8317378.

[40] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, Pisa, Italy, December 2008. IEEE. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.17. URL http://ieeexplore.ieee.org/document/4781136/.

[41] Giang Hoang Nguyen, Abdesselam Bouzerdoum, and Son Lam Phung. *Learning Pattern Classification Tasks with Imbalanced Data Sets.* IntechOpen, October 2009. ISBN 978-953-307-014-8. doi: 10.5772/7544. URL https://www.intechopen.com/books/pattern-recognition/learning-pattern-classification-tasks-with-imbalanced-data-sets. Publication Title: Pattern Recognition.

[42] Elizabeth L. Ratnam, Steven R. Weller, Christopher M. Kellett, and Alan T. Murray. Residential load and rooftop PV generation: an Australian distribution network dataset. *International Journal of Sustainable Energy*, 36(8):787–806, September 2017. ISSN 1478-6451, 1478-646X. doi: 10.1080/14786451.2015.1100196. URL https://www.tandfonline.com/doi/full/10.1080/14786451.2015.1100196.

[43] Solar home electricity data - Ausgrid, 2020. URL https://www.ausgrid.com.au:443/Industry/Our-Research/Data-to-share/Solar-home-electricity-data.

[44] Anaconda | The World's Most Popular Data Science Platform, 2021. URL https://www.anaconda.com/.

[45] 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. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL http://jmlr.org/papers/v12/pedregosa11a.html.

[46] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. *arXiv:1309.0238 [cs]*, September 2013. URL http://arxiv.org/abs/1309.0238. arXiv: 1309.0238.

[47] Mike Kraus. Using Bayesian Optimization to reduce the time spent on hyperparameter tuning, March 2019. URL https://medium.com/vantageai/bringing-back-the-time-spent-on-hyperparameter-tuning-with-bayesian-optimisation-

[48] Gustavo Batista, Ronaldo Prati, and Maria-Carolina Monard. A Study of the Behavior of Several Methods for Balancing machine Learning Training Data. *SIGKDD Explorations*, 6:20–29, June 2004. doi: 10.1145/1007730.1007735.

[49] Erich Schubert, Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems*, 42(3):1–21, July 2017. ISSN 03625915. doi: 10.1145/3068335. URL http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=124428091&scope=site. Publisher: Association for Computing Machinery.