

École polytechnique de Louvain

Anomalies Detection using Ensemble Learning in carbon report data

Author: **Amine MOUNZER**

Supervisor: **Yves DEVILLE**

Readers: **Tanguy DE BELS, Jean-Charles DELVENNE, Alexander GERNIERS**

Academic year 2022–2023

Master [120] in Computer Science and Engineering

Acknowledgments

My first acknowledgments goes to my supervisor, the professor Yves Deville who kindly accepted my thesis proposal and guided my odd work during the whole year.

Furthermore, I would like to thanks my relatives who always supported me and pushed me forward.

Many thanks to Tapio's direction, Louis and Pierre, who accepted this collaboration and were always available when needed. I am also grateful to Tapio's carbon experts, especially Maud and Colin, who took the time to help me on all subjects related to carbon reports and who were always available during the real cases testing.

Lastly, I could not have undertaken this journey without Tanguy, my mentor at Tapio, who guided me during the whole creation of this thesis. The many discussions we had allowed me to explore a lot of directions and helped me building the good basis of this work. The way he challenged me for each part of this project pushed my thinking as far as possible.

Thanks for all your support and encouragements.

Contents

1	Context	5
1.1	Tapio	5
1.1.1	What is Tapio ?	5
1.1.2	Methodology	6
1.2	Problem definition	7
1.2.1	Tapio's Process	7
1.2.2	Problem Statement	8
1.2.3	What is an anomaly ?	8
1.3	Solution: ADTK	10
1.4	Related work	11
1.4.1	Anomalies detection in unsupervised data	11
1.4.2	Ensemble learning	12
1.4.3	Resampling	13
2	Data	15
2.1	Data Retrieval	15
2.2	Data Preparation	16
2.2.1	Data splitting	16
2.2.2	Feature engineering	17
2.2.3	Anomalies generation	17
2.2.4	Feature selection	21
2.2.5	Missing values	22
2.2.6	Scaling	22
2.2.7	Encoding	23
3	Architecture	25
3.1	Ensemble learning	25
3.1.1	Learning	26
3.1.2	Aggregation methods	28
3.2	Individual training	34
3.2.1	Pipelines	34

3.2.2	Algorithms	34
3.2.3	Feature Importance	36
4	Experimentation	38
4.1	Models validation	38
4.1.1	Metrics	38
4.1.2	Basic Tests	39
4.1.3	Tuning	41
4.1.4	Cross Validation	41
4.2	Ensemble block	44
4.2.1	Aggregation techniques	44
4.2.2	Models contribution	49
4.3	Testing and comparison	50
4.3.1	Clustering algorithm	50
4.3.2	Generalization	50
4.3.3	One error per line	50
4.4	Test with real carbon report	51
5	Discussion	54
5.1	Thesis process with Tapio	54
5.1.1	Why Tapio	54
5.1.2	Tapio's goal	54
5.2	Integration in Tapio's process	55
5.2.1	When to use the ADTK	55
5.2.2	Reporting	55
5.3	Limitations	57
5.3.1	Specific anomalies	57
5.3.2	Limited to Tapio's data	57
5.4	Improvements	57
5.4.1	Flexibility	57
5.4.2	Meta-model for aggregation	58
5.4.3	Dispenser's pipeline	58
5.4.4	Proposition	58
5.4.5	Textual anomalies detection	59
5.5	Conclusion	60
	References	62
	Appendices	66

A	Tapio	67
A.1	Feature description of training set	67
A.2	Tapio's driver	67
A.3	Tapio's sectors	69
A.4	Tapio's calculation of uncertainty	69
B	Algorithm details	71
B.1	RandomForest	71
B.2	XGBoost	72

Chapter 1

Context

1.1 Tapio

This master thesis has been achieved in collaboration with the climate tech Start-Up: **Tapio**.

1.1.1 What is Tapio ?

Tapio is a small company of around 20 employees/interns based in Brussels. It was first created in 2019 by Louis Collinet (CEO) and Nathan Clarke. Tapio converts the climate change to a source of inspiration. Elected Start-Up of the year 2021, they stand among the first Start-Up climate tech pioneer in Europe. Their goal is to provide the support that companies need to reach the global carbon neutrality. This is done by combining bio-engineer's expertise and an advanced technology delivered by engineers/computer scientist, in an all-in-one collaborative climate toolbox: tapioview.com.

Tapio works with partners and clients through three distinct "Pillars":

Understand: Tapio collects client's data using their collaborative platform and creates the link between emission and the complete value chain. The final result is a carbon report that offers a deep comprehension of the client's carbon footprint by assessing its CO₂e emissions.

Act: Tapio establishes realistic reduction targets by following the SBTs ¹ and generates a reduction strategy using solutions contained in their databases. They then supports their clients to help them achieving those goals.

Engage: With a lot of transparency, Tapio communicates successes with all stakeholders: suppliers, clients, investors, employees, ... With as purpose the creation of

¹Science Based Targets: <https://sciencebasedtargets.org/>

a climate aware eco-system.

1.1.2 Methodology

Tapio's platform is aligned with the GreenHouse Gas Protocol. It is an international protocol that proposes a framework to measure, count and handle greenhouses gas emissions coming from public and private sector activities elaborated by the World Business Council for Sustainable Development 2 and the World Resources Institute 3. The complete explanation of Tapio's methodology can be founded on their website². In order to have a global understanding of the GHG protocol, this section will, in a nutshell, describe of the most important part of this protocol.

GHG emissions are calculated using the following formula: $activity\ data * emission\ factor = GHG\ emission\ (tCO_2e)$.

The activity data from a company corresponds to the quantity of a specific product or service (e.g., amount of kWh of electricity used, numbers of cars, tons of product etc.). The emission factor is specific to each activity data and therefore allows to convert the activity data into the quantity of CO₂ equivalent emitted (for example: burning 1 litre of petrol=2.3 kg CO₂e). These emission factors are available in various carbon databases such as, among others, the ADEME database or the IPCC.

GHG emissions are then aggregated to calculate the total and disclose the distribution of emissions by **scope** and **categories**.

Scopes are operational boundaries that gathers information about the on-site and off-site activities, processes, services, and impacts to be considered in the carbon report. The objective is to take into account all the sources of GHG emissions on which the company depends to carry out its activities (whether they are direct or indirect). There are three distinct scopes:

- Scope 1: Direct emissions from sources owned or controlled by the organization, such as: combustion of stationary and mobile sources, non-combustion industrial processes, ruminant emissions, refrigerant leaks, biomass, etc.
- Scope 2: Indirect emissions associated with the production of electricity, heat or steam imported for the organization's operations.
- Scope 3: Other indirect emissions produced by the organization's activities that are not accounted for in Scope 2 but which are related to the complete

²<https://www.tapioview.com/methodology/>

value chain such as: business travel, commuting, inputs, waste, inbound and outbound logistics, ...

GHG emissions from the three scopes are themselves divided into 23 **Categories**, allowing a better tracking and identification of emissions hotspots.

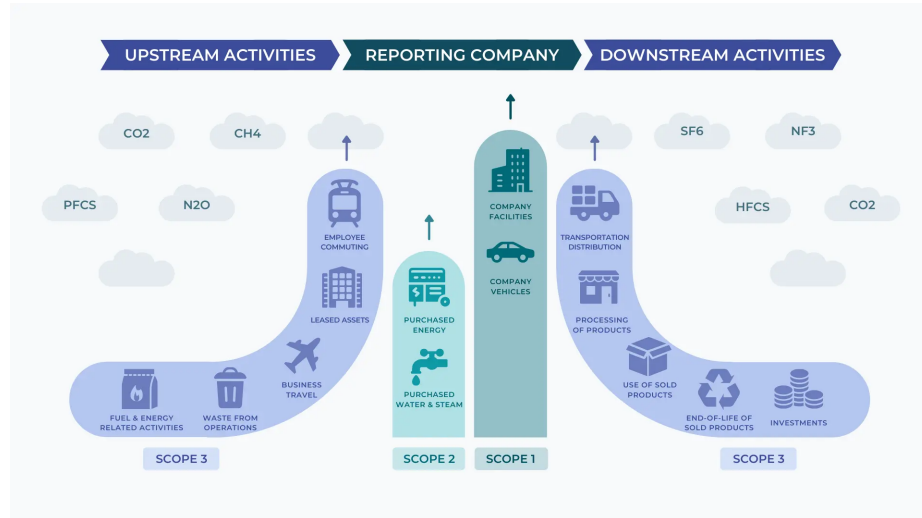


Figure 1.1: GHGs emission repartition in a nutshell (Tapio 2019)

Tapio has based its methodology on this protocol but has considered a specific point that needed to be improved. Categories are hard to read and fully understand. To simplify the comprehension and visualization of a carbon report, Tapio has created “emissions drivers” that regroup emissions sources by “sections” as described in ???. Those drivers are much more human-readable and can be explained by anyone.

1.2 Problem definition

Tapio’s strength is a combination of carbon expertise and advanced technologies that the company owns. They know the impact and importance that new artificial intelligence technologies could have on their tools. After scoping with them the various tasks that such techniques could achieve, we determined the major problem that this thesis could overcome.

1.2.1 Tapio’s Process

In Tapio’s process, the client encode its data in order to generate a carbon report. Those data are concatenation of "rows" representing an unique activity data. In

this record, one could find attributes such as the unit, the category, the driver, etc. Actually, each record of those data has to be verified **row by row** by the carbon expert affected to the project as any misinformation or mistake in the data could much likely lead to a different carbon report, hence wrong targets and reduction.

This makes Tapio's tool fully dependant to carbon expert and prevent from sharing the process to external users such as dispensers³.

In addition, the GHG protocol can sometimes have multiple interpretations, especially in the categories or the emission factor chosen for a specific activity data and this depends highly on the carbon expert experience.

1.2.2 Problem Statement

As the process of data verification is long and mobilizes a full time carbon expert, the goal is to build an anomaly detection tool to be able to find easily and automatically the anomalies that could appear during the collect of the data. This could also lead to a more independent tool, where the carbon expert would only step in when there is a real need for it. Dispensers could use the tool with the insurance that all anomalies will be automatically reported. This would also bring much more coherence between Tapio's carbon expert and in Tapio's whole process.

1.2.3 What is an anomaly ?

In order to solve the stated problem, we need to define exactly what we call an **anomaly**. According to [Chandola et al., 2009], anomalies are data points that do not conform to expected behaviours. Anomalies are often referred as outliers in the machine learning community. Following [Hawkins, 1980], an outlier is "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". In [Xu et al., 2019], we can read that there is no widely acceptable formal definition of this concept. In the literature, an anomaly is also referred to as an outlier, a discordant object, an exception, an aberration, or a peculiarity, depending on specific application scenarios.

However, in the scope of this master thesis, we will make a clear distinction between anomalies and outliers. Indeed, outliers may deviates from observations but could still be correct values while anomalies are absolutely incorrect. In figure 1.2, we can clearly observe this difference between those two concepts.

³Dispensers are external carbon expert that would use Tapio's platform to manage their clients

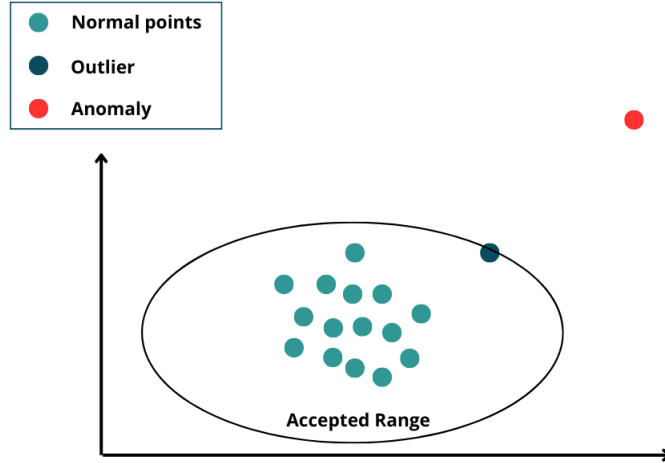


Figure 1.2: Anomaly vs Outlier

It is important to do such clarification when working in a domain where data are highly versatile. The data Tapio are working with are highly diversified and depend a lot on the specificity of their clients. We cannot adequately find anomalies using a simple (deterministic) rule-based solution. A large number of factors could influence the outcome, it is impossible to determine thresholds to detect easily anomalies in such data, therefore we need a more powerful tool. In addition, the goal is to work with large scale carbon report, thus a high number of data. That is why we will use machine learning techniques to classify our data and determine whether it contains any anomaly.

Finally, [Chandola et al., 2009] makes a clear distinction between three categories of anomalies detection that depends on the given data and which also apply to many machine learning techniques.

- *Supervised*: Labels for both normal point and anomalies are available
- *Semi-Supervised*: Data contain only a small amount of labeled rows which are usually only correct (normal) points
- *Unsupervised*: No data is labeled

In this thesis, the data used are semi-supervised. Indeed, as Tapio is a running company, the data available is considered as correct, hence we have no data labeled as anomaly.

1.3 Solution: ADTK

Now that the problem has been defined and the clear need of a machine learning algorithm has been stated, we will detail the solution we designed. The initial approach of our solution is to use a simple binary classification where a correct data is labeled as 0 and an anomaly as 1. We have extended this basic technique to meet our various constraints and issues.

The main goal is to provide a helping tool that will flag rows containing anomalies and delivers enough information to understand quickly where the issue is. Additionally, we want to get a precise reporting that will ease the change of this error. We propose a toolkit composed of multiple specialized *ensemble block* of trained models. The figure 1.3 shows the global architecture of the designed ADTK.

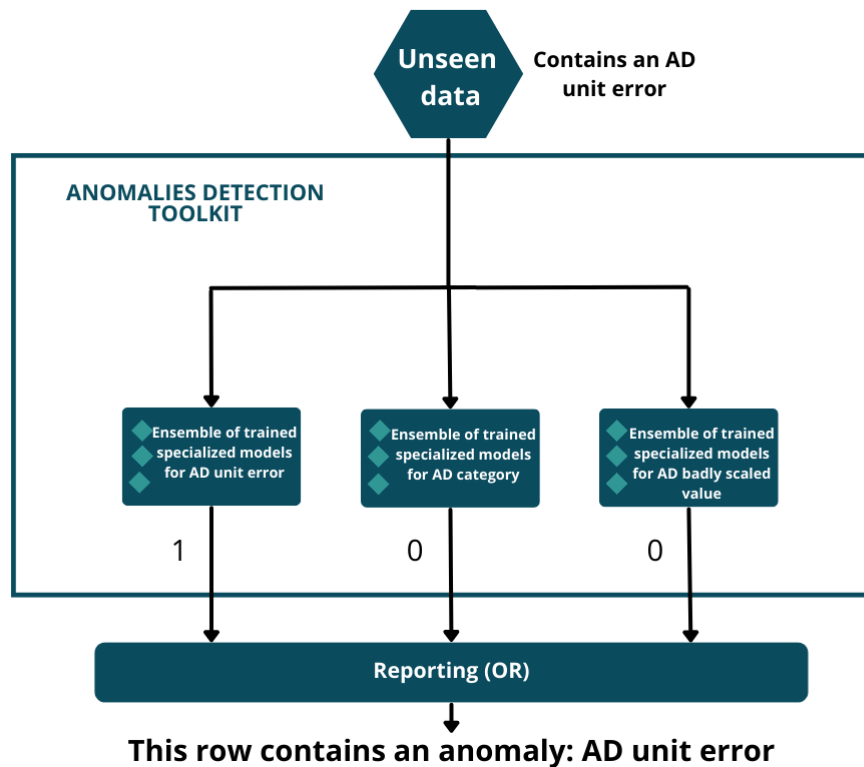


Figure 1.3: Anomalies detection toolkit architecture

When evaluating new unseen data, the toolkit will broadcast it to each of the *ensemble block* and wait for their answer. If one of them outputs 1 then an anomaly has been detected in the data. The ensemble of models are specialized, which means that they contain models (green diamonds on fig 1.3) that have been trained

to detect **specific anomalies**. Therefore, each *ensemble* of models is designed to report an alert (output 1) if it detects that an error for its specific type of anomaly. Obviously, the *ensemble block* will have to give more information, such as the anomaly's row or the associated probability. This will be detailed in the chapter 3.

1.4 Related work

1.4.1 Anomalies detection in unsupervised data

In unsupervised machine learning, data points have no label available, which means that we cannot flag which row is an anomaly nor a normal point. According to [Goernitz et al., 2013], anomaly detection is generally considered an unsupervised task and prominent learning methods. This is because anomalies are rare event and labeling them to train a model is time consuming and could lead to overfitting [Webb, 2010] and bad generalization. The generalization of a model is its ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model. For anomalies detection purpose, it is often considered that it is much more important to characterize the data, finding its pattern or categories to detect when a data point is unexpected.

Anomaly Detection in Temperature Data Using DBSCAN Algorithm

We can show the basic approach of this type of anomalies detection in [Çelik et al., 2011] work. The aim of this study is to discover anomalies on temperature data using DBSCAN, a density-based clustering algorithm. This technique requires two user-defined parameters (hyper-parameters), which are neighborhood distance epsilon (*eps*) and minimum number of points *minpts*. Points are grouped regarding the distance to a centroid point. If this distance is smaller than *eps* then those points are neighbors of the centroid. A group of points is called cluster if it has a number of points greater to the *minpts* parameter. DBSCAN labels the data points as core points (centroid), border points, and outlier points. Border points can be defined as points that are not core points, but are the neighbors of core points. Outliers are points that are not core nor border.

Overall, DBSCAN performs better than a classical statistical approach which will only detect extreme anomalies (above/below threshold). Anomalies are not always such extreme values and can be found by DBSCAN.

1.4.2 Ensemble learning

[Xu et al., 2019] presents an overview of recent progress in the anomaly detection domain. It states that the high-dimensional property of real world data makes them almost equidistant to each other. This implies that any data objects become very close as the dimensionality of data increases, resulting in the meaningless nature of their respective distances. In this case, traditional anomaly detection methods cannot effectively handle high-dimensional data. This article introduces multiple ways to deal with such issues. It especially mentions various forms of ensemble learning techniques developed for the sake of other research.

Feature bagging

The paper [Lazarevic and Kumar, 2005] introduces feature bagging approach, which combines results from multiple outlier detection algorithms that are applied using different sets of features. Each outlier detector identifies different outliers, and thus assigns to all data records outlier scores that correspond to their probability of being outliers. Those individual scores are then combined in order to find the most likely anomalies.

The Outliers detectors algorithm are based on computing the full dimensional distances of the points from one another as well as on computing the densities of local neighborhoods. Each detector will assign its own LOF (Local Outlier Factor = degree of "being an outlier") for each row of a unseen data set. The goal is then to combine the outputs of t detectors into one single LOF vector that will be the final result. To realize this combination, the authors have explored two distinct approaches.

- *Breadth-first* combining: This method sorts all outputs of the individual detectors (by LOF), select the t times the data records with the highest anomaly score from all detectors and finally insert this value into the final output. If the current data exists in the final output, it is not appended again. At the end of we have a final output that contains all indexes of the data sorted according to their LOF among **all** detectors.
- *Cumulative Sum* combining: This method generates the final output by summing all the individual scores, sorting the result and then identifying the highest scores as outliers. This approach enables that an outlier detected by a single detector with a large score may still have a big enough score after the sum to be detected. It thus promotes detectors that have a lot of confidence regarding their output and helps detecting outliers visible only in a few dimensions or by a few detectors.

Mining Outliers with Ensemble of Heterogeneous Detectors on Random Subspaces

In [Nguyen et al., 2010], the aim of the authors is to build an efficient method to combine the results obtained from various outlier detection techniques while overcoming their individual drawbacks when applied on a data set. More precisely, they want to alleviate the curse of dimensionality [Keogh and Mueen, 2017] and noisy features, combine efficiently outlier score vectors of different techniques (and thus different scales/characteristics) and output a higher detection quality than each individual base detectors.

Authors introduce the Heterogeneous Detector Ensemble on Random Subspaces (HeDES) framework.

Each detector will output a vector and those will be stored with other vectors generated by the same type of detector. All sets (each set represents a type of vector) will make a sub-combination to output a single vector. Finally, like in [Lazarevic and Kumar, 2005], the output vectors of each set are combined using a specific function to obtain a final result. Two different techniques are used to do the sub-combination. The adequate technique is chosen based on the set which the vectors belongs to.

- *Weighted Sum* (WS) is used for Ranking-based techniques and will sum all the probability of being an outlier times a weight equal to the performance of the detector. This is similar to the Cumulative Sum proposed in [Lazarevic and Kumar, 2005] but with a weight that helps boost the performances of more efficient detectors.
- *Weighted Majority Voting* (WM) is used for Threshold-based techniques and will do a classical Majority Vote using as weight the performance of the detector.

After that, All assembled vectors from both set will be again combined using an *OR voting*: $V_final = V_ws \vee V_wm$

1.4.3 Resampling

To deal with semi-supervised imbalanced data, many techniques have been explored and may be used.

Sentiment analysis via semi-supervised learning: a model based on dynamic threshold and multi-classifiers

The paper [Han et al., 2020] deals with a totally different problem but solves it in a way that is close to what we aim to do.

The author’s goal is to propose a semi-supervised model based on dynamic threshold and multiple classifier for sentiment analysis. It is a natural language processing (NLP) technique used to determine whether data (text) is positive, negative or neutral. They work with a limited amount of data and since they aim to train multiple classifier to avoid overfitting, they need to artificially generate new coherent data. **Bootstrap sampling** [Kulesa et al., 2015] is used to achieve this task. For each classifier, random samples will be selected and may be reused by other classifier. All samples that were not used to train classifiers will be used for testing purpose.

Afterwards, they combine outputs of each classifier using their performance as weight. This is calculated following this formula: $R_{it}^c = \frac{\frac{1}{F_{it}}}{\sum_{t=1}^T \frac{1}{F_{it}}}$ where R_{it}^c represents the performance of the t th classifier when tested against samples in class i . In this work, $i = 1$ indicates a positive class and $i = 0$ indicates negative class. T represents the total number of classifiers and F_{it} the estimated F-measure value of the t th classifier when tested against the i -class samples.

SMOTE: synthetic minority over-sampling technique

SMOTE [Chawla et al., 2002] is an oversampling approach that deals with imbalanced data set by creating “synthetic” examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k -minority class nearest neighbor.

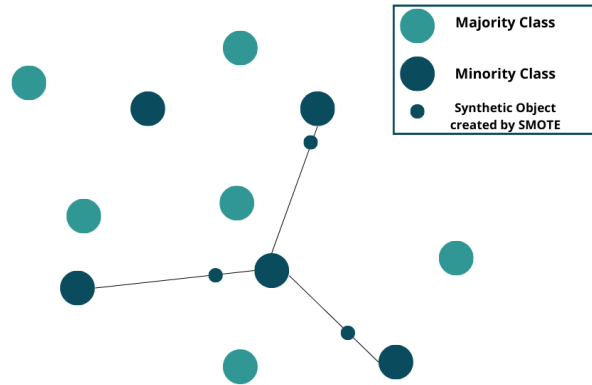


Figure 1.4: Simple SMOTE example

Chapter 2

Data

In each machine learning project, the most important part is handling the data in its entirety. Choosing the pertinent features, retrieving the highest number of records, cleaning them, enriching them to obtain a training efficient data set, etc. The quality of the training set has an enormous impact on the accuracy of the trained models [Sessions and Valtorta, 2006]. In this chapter, we will describe the data used and detail all modifications or enhancements that were made to upgrade it. All steps were done using the sklearn¹ tool in python.

2.1 Data Retrieval

The data collected by Tapio is large. Since the focus of this thesis is to detect anomalies on specific areas, a first step is to select the pertinent features and records among Tapio’s database. The data retrieved is a collection of activity data (ADs) from multiple companies and reports, with various features to describe them. The features choices were made after many discussions with multiple Carbon experts from Tapio. Each AD belongs to a specific reports from a company and is associated with an Emission Factor to calculate its total emission (CO₂eq).

The records were gathered on what Tapio calls *Published reports*. Those are reports that were completed by the client and then verified by Tapio’s carbon experts. They are considered as **correct** report, which guarantee us to work with real world values encoded and used by companies. The original data set on which we worked count 16 features and around 2800 distinct records. Figure 2.1 shows a simplified schema of the data features and their types. Detail definition of those features can be found in table A.1 in the Appendix.

¹<https://scikit-learn.org/stable/>

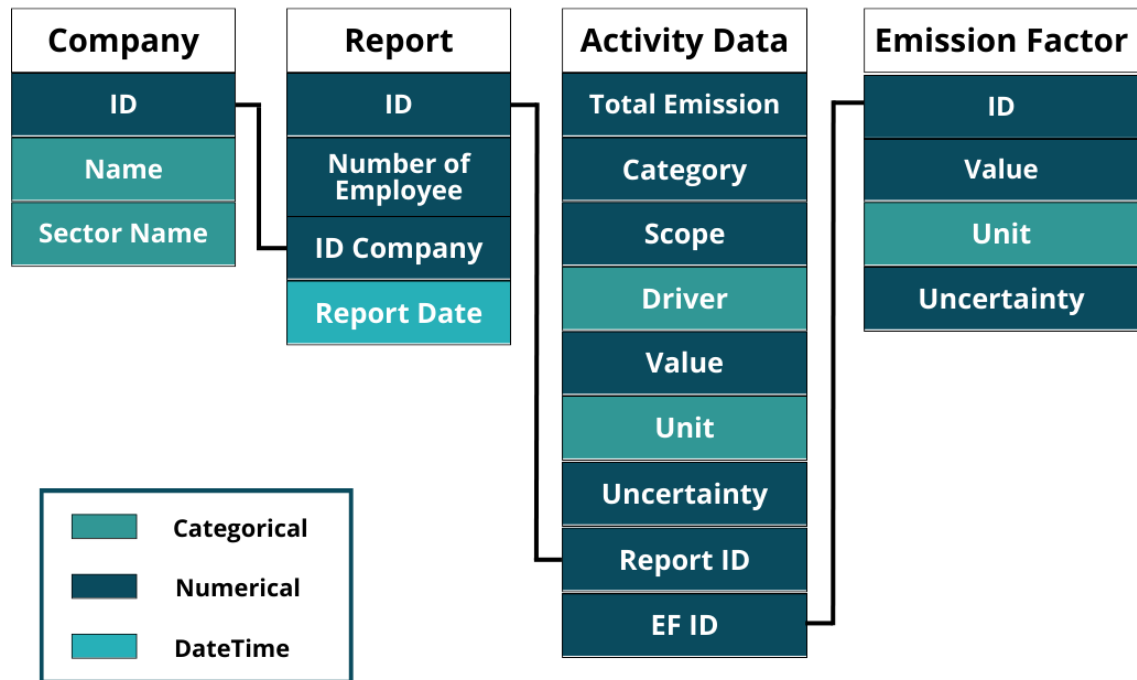


Figure 2.1: Simplified diagram of data used

2.2 Data Preparation

Data preparation, often referred as data preprocessing, is the most time consuming part of a machine learning project. It consists of dozens of transformations and needs to be repeated several times. Despite advances in technologies for working with data, each of those transformations may involve much-handcrafted work [Abdallah et al., 2017]. Nevertheless, there are enough commonalities across predictive modeling projects that are defined as a sequence of steps and subtasks that must most likely be executed [Brownlee, 2020].

This section will detail in depth all modifications that were made on the original data to achieve our goal.

2.2.1 Data splitting

Data splitting is commonly used in machine learning to divide data into a train, test, or validation set. This approach allows to find the model hyper-parameter and estimate the generalization performance [Abe, 2001].

To test our models, we need values that come from real data to attest the good performances of our trained models. Moreover, those data must be unseen, meaning that it should never share values with the sets used to train our models. Hence,

we split the original data in two data set, a train set and a test set (80/20). In addition, a specific issue that we could face is the overfitting [Webb, 2010], where our models would too much rely on what they have learned to make predictions and would have bad generalization. This could happen especially regarding the training companies and reports. To verify overfitting, we remove all ADs of a chosen company from the training set and add it to the test set. In chapter 4, we will analyse and discuss such dependencies in our models.

2.2.2 Feature engineering

Feature Engineering (FE) refers to the process of creating new input variables from the available data. This is highly specific to the given data and data types [Brownlee, 2020].

Adding useful information to our data could increase our performances especially in domain such as detection of anomalies. Two precious features were calculated and added to our original training set.

- **Number of AD in report:** it is the total number of ADs contained in the AD's report. It gives information on the size of the report
- **Proportion of AD's emission in the global report:** calculated as $\frac{AD_{total_emission}}{Report_{total_emission}}$ Where the Report Total Emission is the sum of all Total Emission of each AD composing the Report. This could help the model determining if an AD has too much (or not enough) impact on the report regarding its category, driver, emission factor, ...

2.2.3 Anomalies generation

Before going further in the classical data preparation, we need to add records to our data that will represent the anomaly class. Indeed, as said previously, the data retrieved from Tapio is considered as **correct**, which means that each record is **not an anomaly**. Since our approach to deal with the specified problem is the supervised learning, we need to generate our own anomalies. This is a difficult task for computer scientist since it requires specific knowledge on the given data. After multiple discussion with Tapio's carbon expert team, we were able to determine specific processes to create feasible anomalies that would need to be detected and that can be generated manually. However, it is important to note that such anomalies can still not be founded using rules nor thresholds given the high versatility of carbon report data and that machine learning is necessary to detect automatically such errors.

Creating new data from scratch is not a relevant option. All the data used to train a model has to be coherent to not generate random outliers. Otherwise, it

would be impossible to determine the specificity of given anomalies. Therefore, instead of seeing this as a "lack of data" issue, we define our data set as imbalanced² ([Fernández et al., 2018]) where we have 100% of the first class (correct values labelled as 0) and 0% of the second class (anomaly values labelled as 1). Many techniques exist to deal with such data set by doing resampling. They are divided in two categories, undersampling and oversampling which respectively decrease the number of records from the majority class and increase the number of records from the minority class. Obviously, our goal is to create anomalies and use oversampling but also for multiple other reasons:

- We do not want to lose information about the correct data
- We want to have as many data as possible
- We already have a limited number of data

A possibility to achieve such oversampling has been introduced in the chapter 1 and is called SMOTE, for synthetic minority over-sampling technique. This method allows to generate brand new records that are close to existing ones. However, this technique was not chosen since working with synthetic data could lead to much errors. Indeed, in highly versatile data such as Tapio's set, a minor change could lead to an incoherent data on which we do not have any view, meaning we do not know the origin of the error. Hence, it would be impossible to predict the kind of error detected by models trained with such data.

Instead, we use a well known technique called **Bootstrapping** (figure 2.2). It is a resampling technique that involves repeatedly drawing samples from the original data with replacement, meaning that the same data point may be included in our resampled data set multiple times [Dixon, 2006].

²An imbalanced data set is a classification data set with skewed class proportions. Classes that make up a large proportion of the data set are called majority classes. Those that make up a smaller proportion are minority classes.

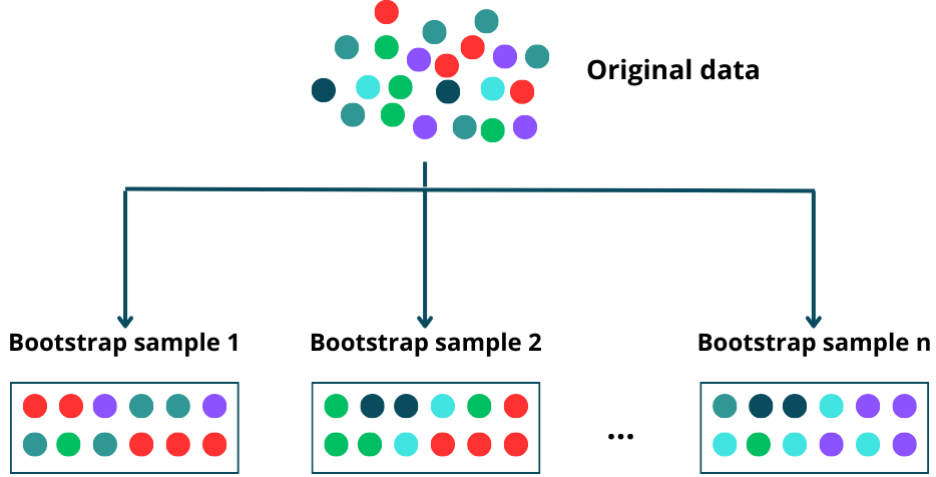


Figure 2.2: Bootstrap sampling example

The records created by bootstrap are coherent, however, to use them as anomalies we need to insert specific errors in them. This is done by modifying a single feature for each record of the bootstrap sample. As said previously, we determined how to change those features in collaboration with carbon experts to create feasible error for a human. The error created in this feature becomes what our ADTK needs to find and describe. For example, if we want to create a data set to train a model that will be able to detect anomalies on the *Activity Data Value* feature, we will execute the following steps:

1. Take half of the original data set randomly and label them as correct values (0)
2. Take the other half, generate $n/2$ bootstrap samples with replacement (where n is the size of the original data set)
3. Change the AD value of each record in the bootstrap sample using a pre-determined formula, then label them as anomalies

However, in practice, this would give us a high number of specified data set that need to be preprocessed, which is computation wise not efficient and redundant. Therefore, when creating a complete training data set, we repeat the step 2 and 3 for each anomaly type that we want to generate. Then, when the model training occurs, a filter will be applied to assign to each model a specific data set with all correct records and all records containing a single anomaly. The schema 2.3 shows how a complete training set is created when working with 2 different type of anomaly.

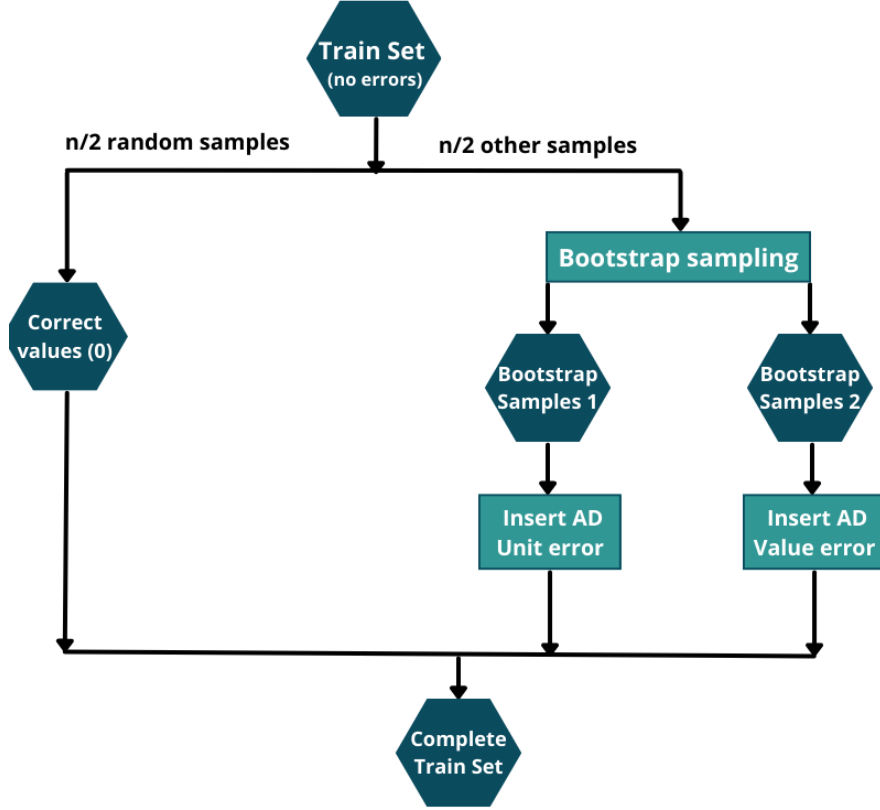


Figure 2.3: Anomalies generation process with 2 different types of error

To distinguish the different anomalies, we add the type of anomaly generated as column. Here is a list of the different anomaly type we have generated and tested after many discussion with Tapio's carbon experts:

- **Scaling Error** on the *Activity Data Value* feature: The values of AD are sometimes poorly filled (due to misunderstanding of the unit, AD or EF). To create the error, we thus multiply the AD value of each record by a random number contained in a established list. For example, the value could be multiplied by 1000 because tons and kg are often confused during the data collection
- **Unit Error** on the *Activity Data Unit* feature: The units are often confused and this is a common source of mistakes at Tapio. It is easy to catch such error manually, but hard to do it automatically because of the versatility of the data. To create the error, we change the unit of each records by a unit of its family (when possible). It means that an AD with the unit *meter* (m) will have its unit converted to cm, dm, km, etc.

- **Category Error** on the *Activity Data Category* feature: The category are strict and represent the type of AD. It happens that an AD is wrongly categorized. To create such error, we randomly change the category of the AD by another one from the 22 available categories.
- **Driver Error** on the *Activity Data Driver* feature: As for the category, AD could be put in the wrong driver. To create the error, we change the driver of each record by a random one among the 7 others.

Finally, all the generated data are gathered together with the correct values in a single data set that will undergo the *preprocessing pipeline*.

2.2.4 Feature selection

Some Features were necessary for previous steps (feature engineering) but should not be included in the final train set. We thus remove the *Company name* and the *Report name* because we want the best generalization possible for our models. Hence, we do not want them to depend on specific names or companies.

When choosing our best features, it is also important to check if some of them are redundant. Indeed, two columns could give more or less the same information to our models, we would thus remove one of them to reduce the number of features. This is called **correlation** [Schiefer and Schiefer, 2021] between features and can be verified using a simple *heat map*. The heat map of our data is displayed in figure 2.4 and shows that almost none of the attributes are correlated, meaning that they may all contribute to the models's training. However, we can observe that the "report total emission" (a feature that we manually added during feature engineering) is highly correlated with the "number of employee" attribute. This can simply be explained by the fact that bigger companies have by default bigger emission. This assumption makes sense but could be wrong in certain close cases, which is why we decide to keep both features.



Figure 2.4: HeatMap for numerical features

2.2.5 Missing values

In real world data, features may be missing at training time. Those missing values are usually attributed to: human error when processing data, machine error due to the malfunctioning of equipment, respondents refusal to answer certain questions, drop-out in studies and merging unrelated data [Tlamele et al., 2021]. It may cause performance degradation, data analysis problems and biased outcomes.

In our case, missing values are only present in two features: AD and EF uncertainty. This is due to the difficulty to collect this value for carbon experts. Those losses do not have a high impact on the data itself nor on the models performance. Hence, we use a simple Imputer technique and replace all missing values by the median of all existing uncertainty for each feature. This approach allows that the choice of the uncertainty will not be biased by outliers, most of the uncertainty values are included between 5 and 20 %.

2.2.6 Scaling

Feature scaling is a technique to standardize independent features in a fixed range. In other words, it brings data points that are far from each other closer.

It prevents machine learning models to weigh greater values higher. In our data, scaling has a great importance since it has a high versatility. Without scaling, models will suffer from the high difference of order of magnitude in the data. Many scaling techniques exist, with their strength and weakness and we need to find the best one for our data [Ahsan et al., 2021].

Standard Scaler

This scaler standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: $z = \frac{(x-u)}{s}$ where u is the mean the training samples.

MinMax Scaler

Transform features by scaling each feature to a given range between 0 and 1. The transformation is given by: $x_{scaled} = x_{std} * +min$ where $x_{std} = \frac{X-min(X)}{max(X)-min(X)}$. Both StandardScaler and MinMaxScaler are sensitive to outliers ³.

Robust Scaler

Robust scaler uses statistics to create robustness against outliers. It removes the median and scales the data according to the quantile range.

Robust Scaler was chosen as the best scaler given all performances of the trained models.

2.2.7 Encoding

Most of the machine learning algorithms require all input and output variable to be numeric, meaning that we need to somewhat convert our categorical features to numbers.

In our case, we need to convert four features: Sector name, driver name, AD unit and EF unit. All of them have records where ordinal relationship does not exist. Therefore, it would not be a good idea to only transform them in number following a fixed ordering (driver1 = 1, driver2=2, ...) because this would create a false difference of importance between the features. We need a technique closer to reality, that is why we use **One Hot Encoding**. It is an encoding method that removes the original feature and adds a binary feature for each unique categorical value.

³Remember that those outliers are not the anomalies that we added into our data. Here, outlier defines a point that is far from other points

id	Colors	One Hot Encoding			id	colors_red	colors_blue	colors_green
1	Red	→			1	1	0	0
2	Blue				2	0	1	0
3	Green				3	0	0	1

Figure 2.5: One Hot Encoding simple example

Each record will thus have the column corresponding to its categorical value at 1 and all other at 0. A simple example can be seen in figure 2.5.

One hot encoding is excellent but has a major drawback, it increases significantly the number of features, which can cause issues for memory and for the curse of dimensionality. However, in our case, we have a low number of features so it is acceptable to use this technique to encode all categorical features. It is important to note that after using One Hot Encoder, our data went from 13 features to 130. This is especially due to the highly versatile possibilities of the two columns **unit**.

Chapter 3

Architecture

This chapter will detail the chosen architecture to solve the given problem. As showed in 1.3 in chapter 1, the Anomalies Detection ToolKit is composed of multiple *Ensemble block* containing models trained to be specialized in the detection of the same anomaly. The final objective is to report found anomalies with additional information such as the type of anomaly and the probability associated.

3.1 Ensemble learning

According to [Zhang and Ma, 2012], *Ensemble learning* is defined as "a way to improve our confidence that we are making the right decision, by weighing various opinions, and combining them through some thought process to reach a final decision".

Working with *ensemble block* of models instead of one specialized model is a realistic choice made regarding the quantity and quality of the data owned by Tapio. We aim to have a robust and stable solution considering the high versatility of the data. Training a single model with one generated data set as explained in chapter 2 could easily lead to overfitting, hence bad generalization. Another downside of this approach is that we only use half of the original data set as correct values while each of the records could be useful to feed a model. Therefore, to gain robustness and stability, we choose to train multiple models for each specific anomalies and combine their result to obtain a final output, the aggregate.

In a nutshell, an *ensemble block* is composed of n models trained to detect the same type of anomalies but with **different** random subset of the original data. Each of the specialized model will thus receive the input and will output for each row a label: 0 if correct, 1 if it considers that the record contains an anomaly.

Afterwards, each *ensemble block* will use an ensembling method (Majority Vote, Highest Probability, ...) to aggregate the models's outputs and determine if the data contains anomalies or not as shown in figure 3.1.

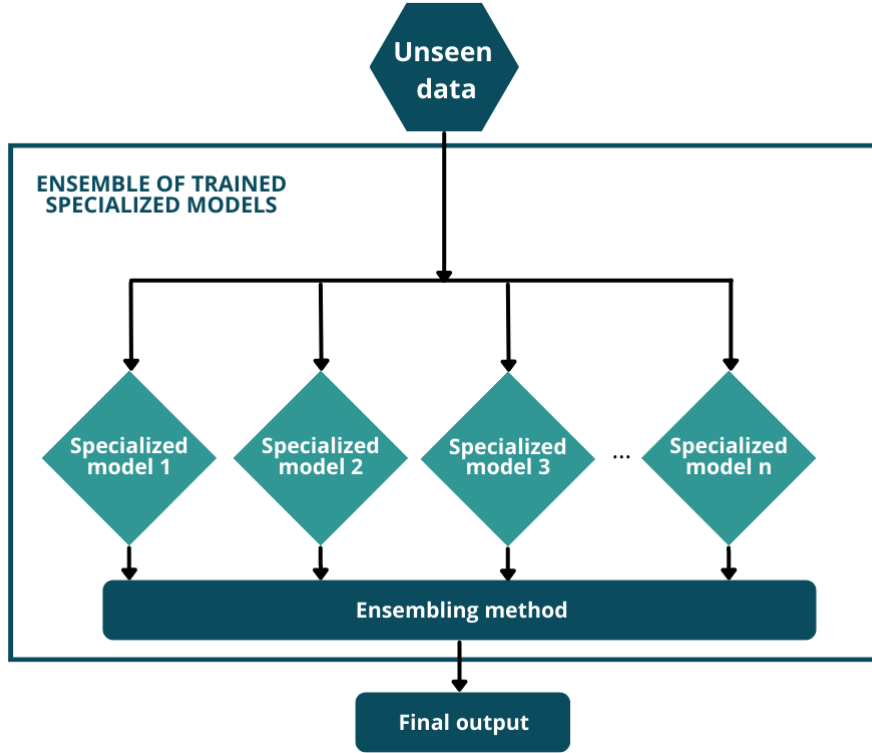


Figure 3.1: Ensemble learning of specialized models architecture

3.1.1 Learning

We need to train all those models with unique data sets to promote specialization and generalization. Starting from a complete training set obtained after bootstrapping and preprocessing as described in figure 2.3, we filter each record to feed a single model for each type of anomaly. Hence, we use each time the rows considered as correct where `anomaly_type = 0` and the rows corresponding to the anomaly we want to detect. For instance, scaling anomalies are labeled as `anomaly_type = 2`. This example is illustrated in the two following figure 3.2 showing the complete data set after bootstrap sampling and figure 3.3 displaying the same set filtered to train only models for scaling anomalies detection.

	nb of employee	total_emission	activity data category	activity data scope	activity data value	activity data unit	emission factor value	emission factor unit	anomaly_type	is_anomaly
0	1343.0	693.36	Fugitive	Scope 1	4.269450e-01	kg	1624.0000	kgCO2e/kg	0	0
1	153.0	12670.00	Stationary Combustion	Scope 1	7.000000e+03	m³	1.8100	kgCO2e/m³	0	0
2531	167.3	59477600.00	Purchased Goods and Services	Scope 3	1.118000e+07	kg (net weight)	5.3200	kgCO2e/kg (net weight)	2	1
2532	20.0	885511.44	Upstream Leased Assets	Scope 3	4.319568e+06	kWh	0.2050	kWh	2	1
3797	96.0	8.87	Fuel and Energy Related Activities	Scope 3	4.927000e+03	kWh ICV	0.0018	kgCO2e/kWh ICV	3	1
3798	496.0	20851.90	Electricity Consumption	Scope 2	1.233840e+05	kWh ICV	0.1690	kgCO2e/kWh ICV	3	1
5063	18.0	608.89	Investments	Scope 3	2.174602e+00	k€	280.0000	kgCO2e/k€	4	1
5064	11.0	5697.15	Mobile Combustion	Scope 2	2.779100e+04	kWh	0.2050	kWh	4	1
6329	50.0	169.00	Electricity Consumption	Scope 2	1.000000e+03	kWh	0.1690	kgCO2e/kWh	5	1
6330	81.0	2740.23	Electricity Consumption	Scope 2	1.336700e+04	kWh	0.2050	kgCO2e/kWh	5	1

Figure 3.2: Minimalist table of complete data set

	nb of employee	total_emission	activity data category	activity data scope	activity data value	activity data unit	emission factor value	emission factor unit	anomaly_type	is_anomaly
0	1343.0	693.36	Fugitive	Scope 1	4.269450e-01	kg	1624.000	kgCO2e/kg	0	0
1	153.0	12670.00	Stationary Combustion	Scope 1	7.000000e+03	m³	1.810	kgCO2e/m³	0	0
2531	167.3	59477600.00	Purchased Goods and Services	Scope 3	1.118000e+07	kg (net weight)	5.320	kgCO2e/kg (net weight)	2	1
2532	20.0	885511.44	Upstream Leased Assets	Scope 3	4.319568e+06	kWh	0.205	kWh	2	1

Figure 3.3: Filtered minimalist table to train only scaling models

Table 3.1 propose a summary of the anomaly_type mapping.

Model	Anomaly Type	Name
SE1	0 & 2	Scaling Error Model
UE1	0 & 3	Unit Error Model
CE1	0 & 4	Category Error Model
DE1	0 & 5	Driver Error Model

Table 3.1: Summary of anomaly type filtering for each specialized model

We repeat the process of figure 3.4 n times using different complete training set, all generated by the bootstrap sampling process. The individual training of a model will be describe in the section 3.2.

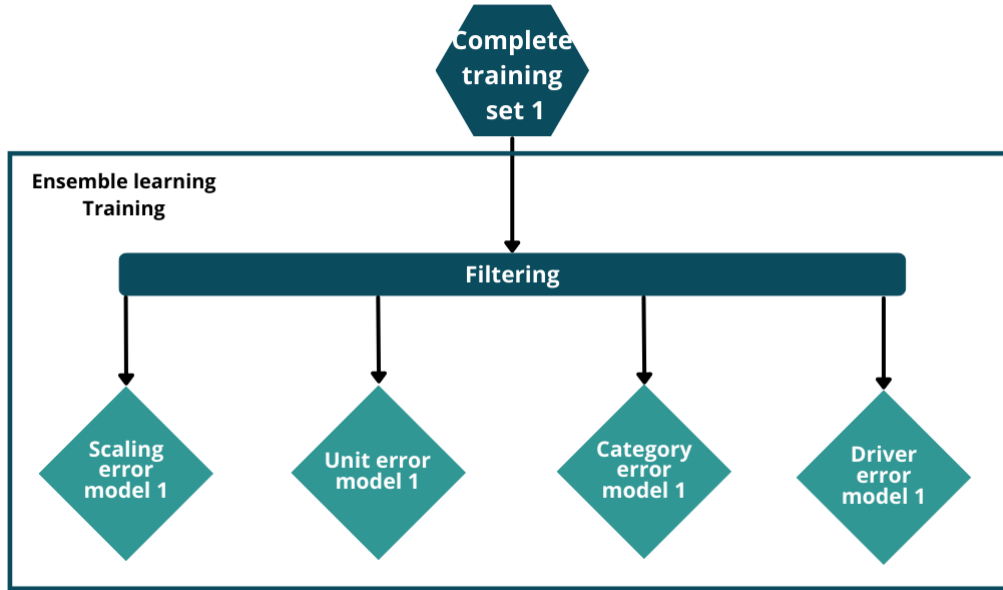


Figure 3.4: Ensemble learning using first generated data set

3.1.2 Aggregation methods

Once all specialized models are trained and tested, we group them by anomaly type in *Ensemble blocks*. For a given input, each blocks uses an aggregation method to combine all outputs of its specialized models. More specifically, to each record, every model will assign a class (0 or 1) and the aggregation method will group those results to output an **aggregate**. There are several approach to combine models's output as we have seen in the chapter 1.4 and has described in [Zhang and Ma, 2012].

Majority Vote

Majority Vote in a binary classification problem is straightforward. For a given input, each model predicts label 0 or 1 and the most picked up class is chosen as the aggregate. The algorithm 1 shows the behaviour of majority vote for a single line x as input

Majority Vote shows high performances when the models's outputs are purely independent. It also helps preventing local maxima by taking multiple outputs into

Algorithm 1: Majority Vote

Data: *single line input : x*

Result: an aggregate that combines all models's outputs using Majority Vote

```
1  $y \leftarrow all\_models.predict(x)$ 
2  $counter\_0 \leftarrow 0$ 
3  $counter\_1 \leftarrow 0$ 
4 foreach  $label \in y$  do
5   | if  $label=0$  then
6   |   |  $counter\_0 \leftarrow counter\_0 + 1$ 
7   | else
8   |   |  $counter\_1 \leftarrow counter\_1 + 1$ 
9   | end
10 end
11 if  $counter\_0 > counter\_1$  then
12 |    $return\ 0$ 
13 else
14 |    $return\ 1$ 
15 end
16
```

account. However, This method does not promote individual strength, meaning that even a model with 99% accuracy may be ignored.

Highest Probability

Each prediction coming from a model goes along with a probability that represents the confidence that this classifier has determined for its answer (numerical value between 0.0 and 1.0). We could make the assumption that the model with the highest probability has better luck to be correct. In the Highest Probability approach, we take as aggregate the output of the model with the highest probability associated.

This method promotes the individual strength of specialized models but works only when all models's performances are good. In addition, it increases the chance of overfitting by relying on one single model.

Algorithm 2: Highest Probability

Data: *single line input : x*

Result: an aggregate that combines all models's outputs using Highest Probability

```
1  $y \leftarrow all\_models.predict(x)$ 
2  $y\_proba \leftarrow all\_models.predict\_proba(x)$ 
3  $max\_index \leftarrow get\_index(max(y\_proba))$ 
4 return  $y[max\_index]$ 
```

Soft Voting

Soft Voting is a method that combines the previous presented strategies by summing all probabilities of each class and taking the one with the highest value.

Algorithm 3: Soft Voting

Data: *single line input : x*

Result: an aggregate that combines all models's outputs using Soft Voting

```
1  $y \leftarrow all\_models.predict(x)$ 
2  $y\_proba \leftarrow all\_models.predict\_proba(x)$ 
3  $proba\_0 \leftarrow 0$ 
4  $proba\_1 \leftarrow 0$ 
5 for  $i \leftarrow 0$  to  $n$  do
6    $label \leftarrow y[i]$ 
7    $proba \leftarrow y\_proba[i]$ 
8   if  $label=0$  then
9      $proba\_0 \leftarrow proba\_0 + proba$ 
10  else
11     $proba\_1 \leftarrow proba\_1 + proba$ 
12  end
13 end
14 if  $proba\_0 > proba\_1$  then
15   return 0
16 else
17   return 1
18 end
```

This approach is recommended if classifiers are well calibrated and solves drawbacks of both previous methods. However, this techniques only rely on the accuracy score which is the metric that evaluates the number of records correctly classified during

training. For our specific problem, we need specific metrics to prevent our tool missing anomalies.

Custom Voting

The evaluation of a model depends on the chosen metrics. For a classifier, this metric is often the number of value correctly classified divided by the total number of elements (accuracy score). However, in binary classification, widely used metrics are **Recall** and **Precision** ([Ting, 2010] and [Melo, 2013]) because they allow to adapt to the kind of problem one wants to solve. To define such metrics, some key concepts need to be clarified.

- **True Positives (TP)**: All records correctly classified as positive
- **True Negatives (TN)**: All records correctly classified as negative
- **False Positives (FP)**: All records wrongly classified as positive
- **False Negatives (FN)**: All records wrongly classified as negative

We may now define precision and recall metric for the positive class.

- **Precision**: total number of documents retrieved that are relevant/Total number of documents that are retrieved. In Machine learning and for the positive class more precisely, one can define it as True positives/Total number of positives predicted ($TP/TP+FP$)
- **Recall**: Total number of documents retrieved that are relevant/Total number of relevant documents in the database. In Machine learning and for the positive class more precisely, one can define it as True positives/Total number of actual positives ($TP/TP+FN$)

Following the figure 3.5, we establish the formulas of Precision and Recall for both classes (correct value and anomalies). We set our correct values (0) as positive and anomalies (1) as negative.

$$\begin{array}{ll} prec_0 = \frac{TP}{TP+FP} & prec_1 = \frac{TN}{TN+FN} \\ rec_0 = \frac{TP}{TP+FN} & rec_1 = \frac{TN}{TN+FP} \end{array}$$

Precision-Recall formulas for both classes



Figure 3.5: Precision and Recall general formula

In a nutshell, we focus on Precision if the problem needs the prediction to be as correct as possible. Otherwise, if we want to spot as many specific value as possible, we optimize Recall. In anomaly detection, the purpose is to be sure to spot any existing anomaly, even if it means flagging too many records, which is equivalent to classify some correct values as anomalies. In our case, we want to have the best Recall for anomalies and, if possible, a good precision for correct values.

The goal is thus to impact the chosen output during aggregation by taking into account the Precision-Recall metrics for each model. Therefore, we add a custom weight for each model to the Soft Voting technique equal to:

$$weight_i = prec_{0i}^{\alpha} * rec_{1i}^{\beta}$$

Where i represents the model id, meaning that each model will use a single weight value determined during the tuning process. The alpha and beta variable will be fixed during the testing process in chapter 4.

Algorithm 4: Custom Voting using Precision-Recall

Data: *single line input : x*

Result: an aggregate that combines all models's outputs using Soft Voting

```
1  $y \leftarrow all\_models.predict(x)$ 
2  $y\_proba \leftarrow all\_models.predict\_proba(x)$ 
3  $precision\_0 \leftarrow all\_models.predict\_precision\_0(x)$ 
4  $recall\_1 \leftarrow all\_models.predict\_recall\_1(x)$ 
5  $proba\_0 \leftarrow 0$ 
6  $proba\_1 \leftarrow 0$ 
7 for  $i \leftarrow 0$  to  $n$  do
8    $label \leftarrow y[i]$ 
9    $proba \leftarrow y\_proba[i]$ 
10   $weight \leftarrow (precision\_0[i] * recall\_1[i]) * *2$ 
11  if  $label=0$  then
12     $proba\_0 \leftarrow proba\_0 + proba * weight$ 
13  else
14     $proba\_1 \leftarrow proba\_1 + proba * weight$ 
15  end
16 end
17 if  $proba\_0 > proba\_1$  then
18    $return\ 0$ 
19 else
20    $return\ 1$ 
21 end
```

All introduced aggregation methods will be tested and covered in the chapter 4. The Custom Voting seemed to be the most promising regarding our problem statement. However, in practice the choice of the combining method depends on the results obtained and an advantage of our architecture is that each *Ensemble block* may work with different aggregation techniques.

3.2 Individual training

This section describes the training process for a single model and details what algorithm could be chosen to perform anomaly detection.

3.2.1 Pipelines

The process to train a single model uses the latest technologies from the **sklearn** ¹ package. This includes the usage of *pipelines* for every classifier. Machine learning pipelines implements and formalize the processes to accelerate, reuse, manage and deploy machine learning models. An automated, reproducible pipelines reduces all effort required to test and deploy a model [Hapke and Nelson, 2020].

Our pipelines contain two big blocks: the preprocessing part that includes all data enhancements detailed in chapter 2 and the modelling part that includes the chosen model instance with a specific algorithm and its parameters (tuned or not). This methodology allows to generate different types of preprocessing or modelling and insert them in our process without repeating the same process or losing previous techniques or informations.

3.2.2 Algorithms

Each model of our architecture is trained independently and this could be done using any Machine Learning algorithm. many algorithms have an affinity with anomaly detection. Hence, an overview of the best techniques for this specific domain is necessary. The paper [Omar et al., 2013] describes many available supervised algorithm.

K-Nearest Neighbor

KNN is a supervised learning technique which uses proximity to make predictions about the grouping of a single data point. In other word, it makes the assumption that close points have the same behaviour. The parameter k determines the number of neighbors on which the algorithm classifies the given point. The class that has the most neighbors is chosen for a single point, following the majority vote technique.

KNN is a widely used algorithm for anomaly detection since the distance it calculates between points can be seen as a local density estimate, an "outlier score". Low local density points regarding the k nearest neighbor are considered by this

¹<https://scikit-learn.org/stable/>

techniques as anomalies and can easily be flagged.[Ramaswamy et al., 2000]

KNN could perform correctly against our problem statement since points that are close in terms of distance should roughly have the same behaviour. However, since the distance is the only criteria, some unusual data that are correct could be classified as outliers. Hence, in our highly versatile data set, KNN could misclassify a lot of records.

Support Vector Machine

The objective of a SVM is to find the hyperplan that has the maximum margin in an n-dimensional (n is the number of features) space that separates all data points. Therefore, in binary classification, the hyperplan gives the ability to directly classify all points to a class. Support vectors are points that are close to the hyperplan and influence its orientation and position.

SVM is a popular algorithm for binary classification since it has the ability to separate the data by not only relying on the distance between each point. It is robust against outliers, meaning that the SVM will be able to differentiate anomalies from correct values even if they are close regarding some features. However, to work appropriately, the SVM technique needs to be able to find a hyperplan, which is not always possible depending on the data set. Regarding our highly versatile data, it might be hard for this kind of technique to create a good enough separation between the data.

Decision Trees

A decision tree is a tree-based technique in which any path beginning from the root is described by a data separating sequence until a Boolean outcome at the leaf node is achieved [Jijo and Mohsin Abdulazeez, 2021]. Decision trees make predictions by following the decisions in the tree from the root node.

Decision trees have several advantages that makes them suitable for supervised anomaly detection [Jalal and Ezzedine, 2020]. In particular, they have easily interpretable structure and are less susceptible to curse of dimensionality. According to [Douiba et al., 2022], many tree-based algorithm have shown their efficiency for anomaly detection such as XGB, lightGBM, Random forests.

Trees should have good performances against our data since they will be able to split the data in optimized way regarding our different features. However, trees often lead to overfitting, which explain why bagging is a commonly used technique

with them. Hence, our ensemble learning architecture should match perfectly decision trees algorithms, especially boosting trees such as XGB or lightGBM that are considered to be strong for anomaly detection [Ikram et al., 2021].

Flexibility of architecture

The designed solution that will be presented in the next chapter offers a lot of flexibility on the choice of the machine learning algorithm as well as the aggregating technique. We enumerated numerous possibilities mainly because we could use each of them for handling different anomalies or *ensemble block*.

3.2.3 Feature Importance

An important factor when creating models in machine learning is on which features it relies to make predictions. This also gives information on how the model is able to generalize given new unseen input. **Feature Importance** is a way to verify which features are the most important for a specific model. The feature importance used here is related to the XGBoost algorithm. It is calculated as the average information gain over all trees, where the gain is, for a given node, the subtraction of its parent node impurity and its child nodes impurities. Those impurities are determined by verifying whether a feature split point improves the performance which is calculated using the Gini purity metric [Li et al., 2020].

Since we use *Ensemble Blocks* hence, multiple models, we want them to generalize in different ways so that they are all able to contribute to the aggregate. Therefore, we want them to have various features on which they base their predictions.

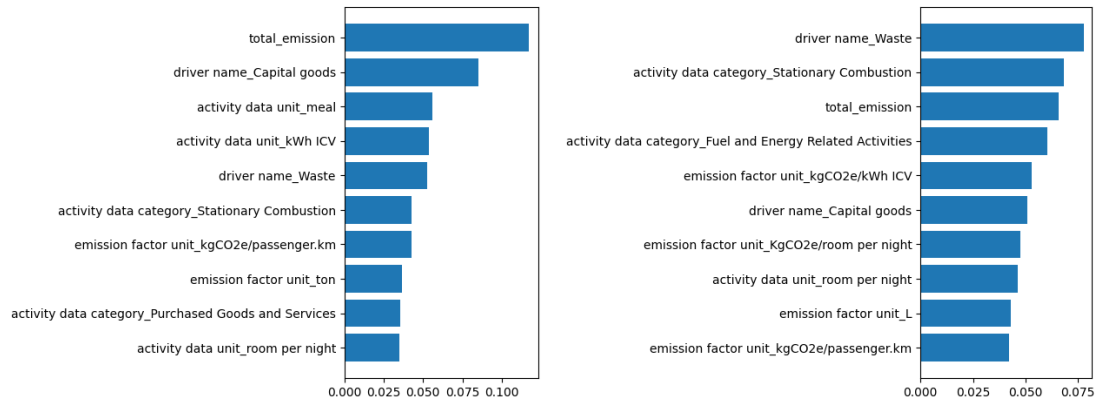


Figure 3.6: Feature Importance of models from the same *Ensemble Block*

In figure 3.6 we display the top ten features regarding their importance for two distinct models trained for detecting the same anomaly but using different training set. XGBoost training algorithm was chosen and was that some essential attributes such as `total_emission`, `driver_name_Waste`, `ad_category_Stationary_Combustion` are in common but not with the same order of importance. Some other features are totally unique to a single model, which reinforces the idea that our models generalize in different manner and will be able to all contribute depending on the given input.

Chapter 4

Experimentation

This chapter describes all experiments that were conducted to verify the performances and quality of the Anomalies Detection ToolKit (ADTK). This process has been realized in collaboration with Tapio's carbon experts during and has been spread out over a long period. Indeed, every part of the architecture needed to be tested and optimized leading to many granular tests that require much time.

In most of the cases, the driver and unit type of anomalies have the same behaviour than the scaling anomaly meaning that their respective results are close to each other. Therefore, most of the time, only scaling and category anomalies testing and results are shown.

4.1 Models validation

This section presents the tests directed to determine what algorithm suits the best to each anomaly. This includes basic testing but also tuning the best algorithms found.

4.1.1 Metrics

To evaluate our model we need to strictly define the metric we will be using. The most interesting measures for our purpose are the precision, recall and accuracy. The precision and recall were defined in the previous chapter, however it is important to note that in the following experiments, we focused on the precision and recall of the anomalies (labeled as 1). Hence, their formulas are:

- $precision = prec_1 = \frac{TN}{TN+FN}$

- $recall = rec_1 = \frac{TN}{TN+FP}$

The accuracy measure is the basic metric used for classification. It is defined as the number of correct predictions made by the model divided by the total number of predictions made.

- $accuracy = \frac{TP+TN}{TP+FP+TN+FN}$

For the smooth running of all testing processes, a specific random state (or seed) has been set everywhere we could to be able to always recreate the same behaviour and not be subject to random effects.

4.1.2 Basic Tests

To make the comparison easier and clearly show the evolution of our testing, we display a baseline performance table as a first step. To do so, we generated 10 different version of the testing set. By different, we mean that the anomalies process creation is not the same for each set but one can find each anomaly type in every set. Therefore, two data sets may share or not the same row, but with a different anomaly or with a different change for the same anomaly.

For a single set and a single anomaly type, we measure the three metrics for each specialized model and calculate the mean of all performances. We then compute again the mean but for each set. We tested this performance measurement with four different algorithms that were previously introduced.

	XGBoost	K-Nearest-Neighbor	Random Forests	Support Vector Machine
Metric				
Recall	0.944037	0.851376	0.936086	0.024465
Precision	0.966839	0.919707	0.968358	1.000000
Accuracy	0.953537	0.882797	0.950322	0.487138

Figure 4.1: Basic performances on test set for scaling anomalies

We may directly observe that Support Vector Machine (SVM) struggles a lot as expected mainly because it is hard to find a hyperplan that properly separates the data. It is especially true for scaling errors where this method labels the majority

	XGBoost	K-Nearest-Neighbor	Random Forests	Support Vector Machine
Metric				
Recall	0.957483	0.461224	0.972109	0.260884
Precision	0.976401	0.687118	0.969494	0.625498
Accuracy	0.967233	0.626486	0.970798	0.509508

Figure 4.2: Basic performances on test set for category anomalies

of the test records as correct values, giving a 100% precision but almost 0.0% recall

K-Nearest-Neighbour has good performances for scaling anomalies but is very bad for category anomalies. This highlights the fact that nearby data points may not share the same behaviour regarding their category.

We see clearly that decision trees based algorithms outperformed other algorithms. XGB uses boosting to render the optimal tree while random forest take the mean of many normal trees trained in different manner. However, all algorithms share a single characteristic. The recall is the lowest value, which is opposed to our goal. As a reminder, we would prefer to flag too much anomaly even if it's wrong than the opposite, we thus want our recall on anomalies to be high.

In the next sections we will detail every technique we used to improve our performances, especially our recall on each model.

Increase recall

The recall metric was lower specifically when testing scaling and unit anomalies. A reason is that the training set was imbalanced and contained more correct records than errors. Indeed, when creating those type of anomalies, we only modify elements that follow a given logic. For example, if we find a record with "kg" as unit, we may create a scaling error by multiplying or dividing the value by 1000 to obtain respectively "g" or "tons". However, at some point, our algorithm encounter odd unit such as "passenger/km". Hence, we cannot follow any rules to convert such records into an anomaly and thus we ignore them. This creates an imbalance data set with as minority class the anomalies and favour the recall of correct values.

To deal with such situation, we could use the **SMOTE** algorithm to oversample the anomalies but we want to control the modification of our data. Therefore, our

simple solution to modify this behaviour is to apply the bootstrapping process until we get the same amount of erroneous records than of correct one. We observe that the recall on scaling anomalies has been upgraded without downgrading the rest of the performances.

	XGBoost	K-Nearest-Neighbor	Random Forests	Support Vector Machine
Metric				
Recall	0.952281	0.889123	0.953860	0.031930
Precision	0.978737	0.927547	0.979645	1.000000
Accuracy	0.954913	0.881156	0.956532	0.362081

Figure 4.3: Upgraded performances on test set for scaling anomalies

4.1.3 Tuning

All presented results of individual models were collected without tuning the available hyper-parameters for each algorithms. However, using tuned parameters boosts our performances in any chosen way. As explained previously, we want to have the best recall for anomalies. Hence, our optimization tuning will focus on increasing the recall performance metric of anomalies. It is important to evaluates the tuning of our parameter without using the test set, otherwise we may add bias in our metrics and our models could perform only for the specific used test set. This is why Cross Validation using validations sets is an essential step in the tuning process.

4.1.4 Cross Validation

Cross-Validation evaluates model by dividing the training set into two subsets, an effective training set and a validation set. As shown in figure 4.4, an example would be 5-fold cross validation where the data is partitioned into 5 equally sized segments. Subsequently 5 iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining 4 folds are used for training [Refaeilzadeh et al., 2009]. To keep a well partitioned data set, we use Stratified KFold¹ that ensures balance in terms of classes between each segments.

¹https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

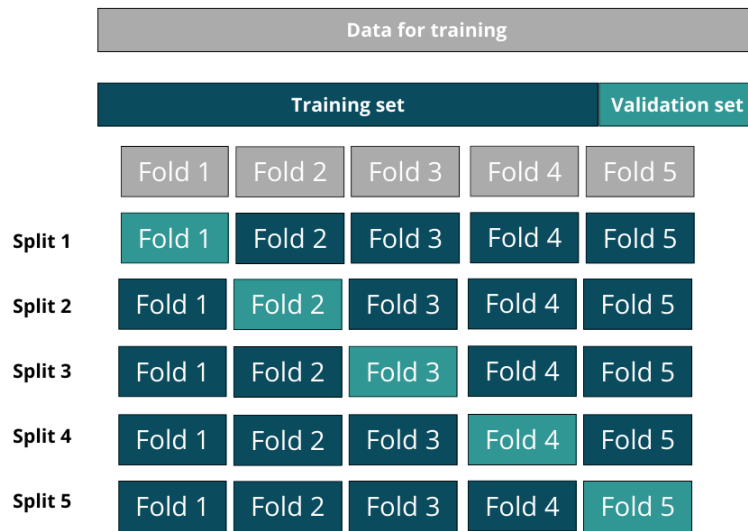


Figure 4.4: Cross Validation process

We run the 3-fold Cross Validation to evaluate our basic model. The displayed result for each accuracy is the mean of 10 runs using differently generated sets on the mean of all specialized model’s performances for their specific anomaly type.

	XGBoost	K-nearest-neighbor	Random Forests	Support Vector Machine
Metric				
Recall	0.973803	0.902981	0.972538	0.025474
Precision	0.969865	0.870525	0.969302	1.000000
Accuracy	0.971725	0.884192	0.970822	0.512737

Figure 4.5: 3-Cross Validation comparison for scaling errors

	XGBoost	K-nearest-neighbor	Random Forests	Support Vector Machine
Metric				
Recall	0.951738	0.510664	0.953791	0.354897
Precision	0.974914	0.704865	0.962866	0.548258
Accuracy	0.963586	0.648065	0.958412	0.508333

Figure 4.6: 3-Cross Validation comparison for category errors

Many methods make use of Cross Validation to tune hyper parameters and upgrade models performances. The one we chose is the most known and used, **GridSearch**.

GridSearch

GridSearch takes predefined parameters values and tries all combination and evaluates the model of each of them using Cross-Validation[Jiménez et al., 2007]. The parameters combination that performs the best following the given metric are then saved.

In the presented architecture, each model uses a single algorithm and is trained with an unique data set. To be optimal, each of them needs to be optimized using a single GridSearch because the best parameter for one model could be the worst for another.

However, testing all possible combinations is computationally expensive and would take a lot of time. Therefore, in this work we make use of **RandomizedGridSearch** which randomly select combinations n_iter times. This does not give an exhaustive search meaning that we do not have the guarantee to get the best parameters. However the gain of performance using classical GridSearch is relatively low regarding the computation time[Bergstra and Bengio, 2012]. We run this test by following the same process as for the basic Cross Validation to be able to compare them without bias.

We observe in tables 4.7 and 4.8 that the recall metric increase for all anomaly type. XGBoost algorithm is slightly better for scaling errors as well as driver and unit errors (not shown). However Random Forests have a 2% mean increase for the recall metric and outperforms XGB for category errors. This demonstrates the flexibility of our architecture, each *ensemble block* could work with a different algorithm. Therefore, we choose to keep models trained with XGBoost for driver, unit and scaling *ensemble blocks* and we choose random forests for category *ensemble block*.

	XGBoost tuned	XGBoost	Random Forest tuned	Random Forest
Recall	0.978	0.974	0.976	0.972
Precision	0.970	0.969	0.971	0.969
Accuracy	0.974	0.972	0.973	0.971

Figure 4.7: Tuned Algorithms performances comparison for scaling errors

	XGBoost tuned	XGBoost	Random Forest tuned	Random Forest
Recall	0.955	0.952	0.970	0.954
Precision	0.973	0.975	0.964	0.963
Accuracy	0.964	0.963	0.966	0.958

Figure 4.8: Algorithms performances comparison for category errors

4.2 Ensemble block

This section will detail the testing conducted to define which aggregation technique would perform the best using our tuned algorithms.

4.2.1 Aggregation techniques

Custom Voting weights tuning

As explained in chapter 3, the **Custom Voting** technique uses weights that represents the recall of anomalies and the precision of correct values for each model. This weights is calculated as follow:

$$weight_i = prec_{0i}^{\alpha} * rec_{1i}^{\beta}$$

Where i represents a single model id. Recall and precision values for each individual model are calculated during the tuning process using the Cross validation outputs. Afterwards, they are stored in a file to be reused when the Custom Voting method is invoked. The alpha and beta parameters need to be tuned in order to get the best custom performances. We make several graphs, fixing each time the value of alpha, going over multiple values of beta and showing the 3 performances metrics: precision_0, recall_1 and accuracy.

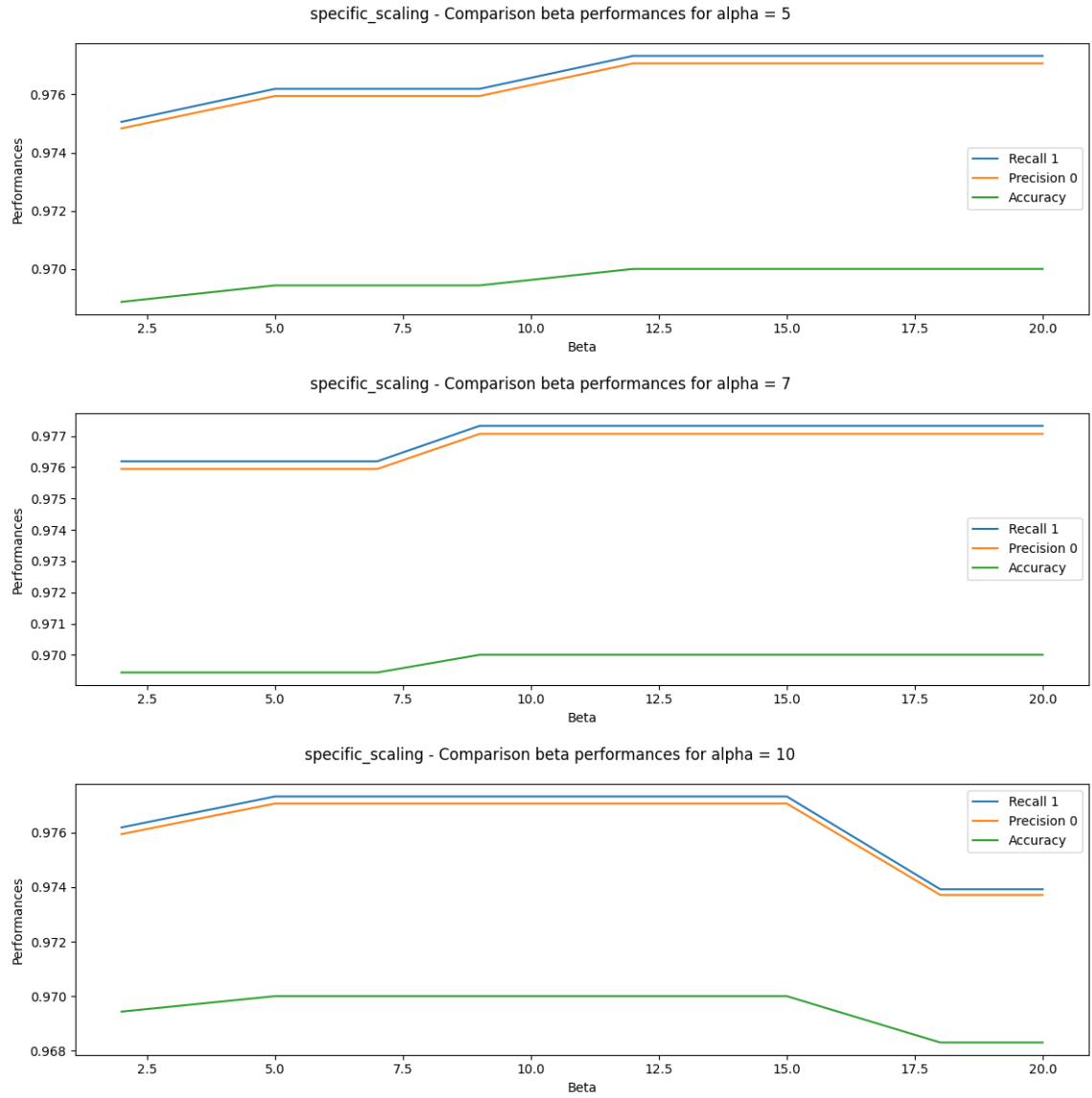


Figure 4.9: Alpha-Beta comparison performances for scaling anomalies

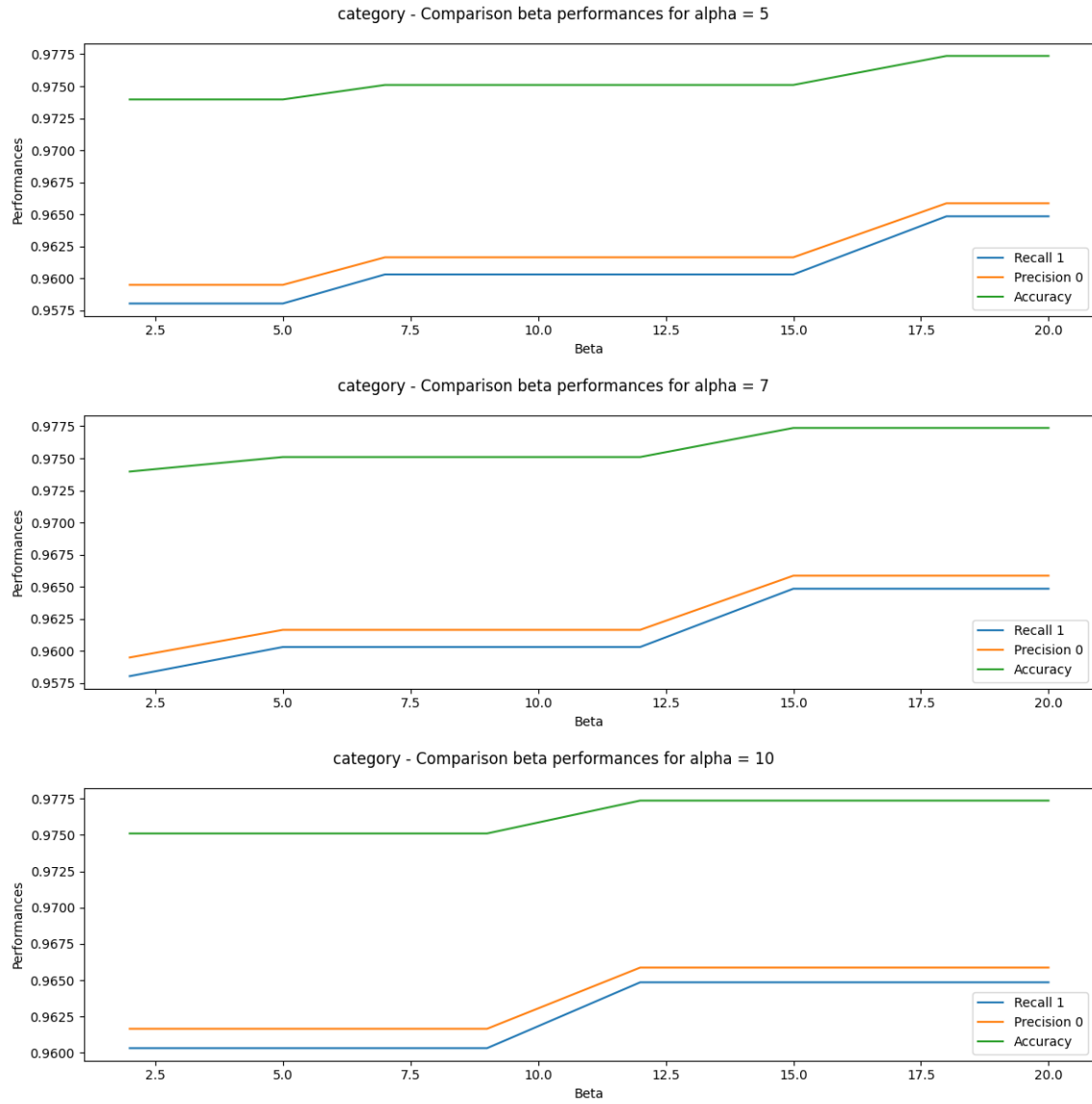


Figure 4.10: Alpha-Beta comparison performances for category anomalies

We notice in figure 4.9 and 4.10 that high values for beta renders better performances, especially for lower alpha. Increasing both parameters above 10 gives worse results in all metrics for scaling. Alpha=7 and beta=10 is our best performances where a plateau is reached. However, for category anomalies we reach the best results and a plateau for alpha = 15 and beta = 20. Therefore, we conclude that each type of anomaly should have a specific Custom Voting formula. The table 4.1 shows the chosen value of alpha and beta for each type of anomaly. This weights tuning allow us to gain globally 1% for each *ensemble block*.

Type of Anomaly	Alpha	Beta
Scaling	7	10
Unit	5	10
Category	15	20
Driver	3	5

Table 4.1: Alpha and Beta values for each anomaly type

Ensemble block performances

We may now use our tuned models and weights to test our ensemble performances with the different aggregation techniques.

Again, the following performances were computed as the mean score over 10 runs using different variation of the test set. The models used are the most performing after tuning.

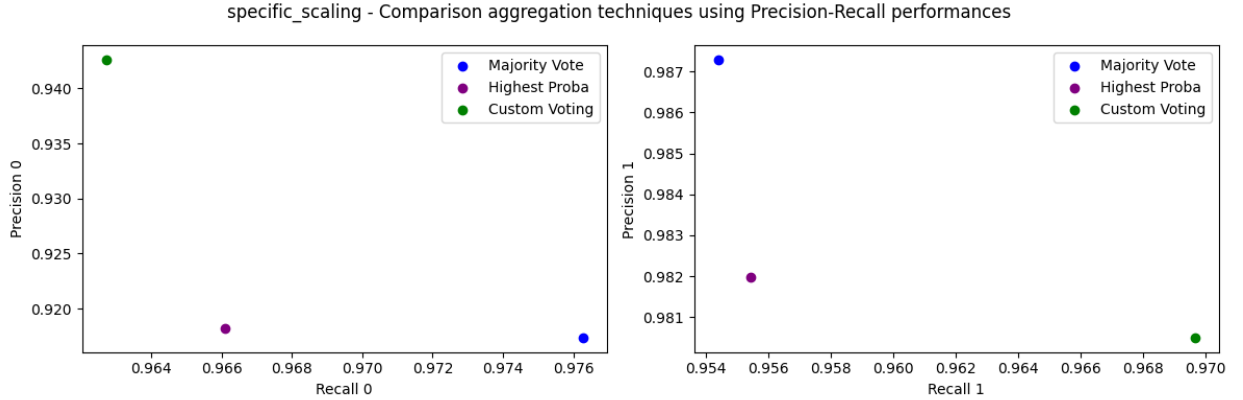


Figure 4.11: Aggregation comparison performances for scaling anomalies

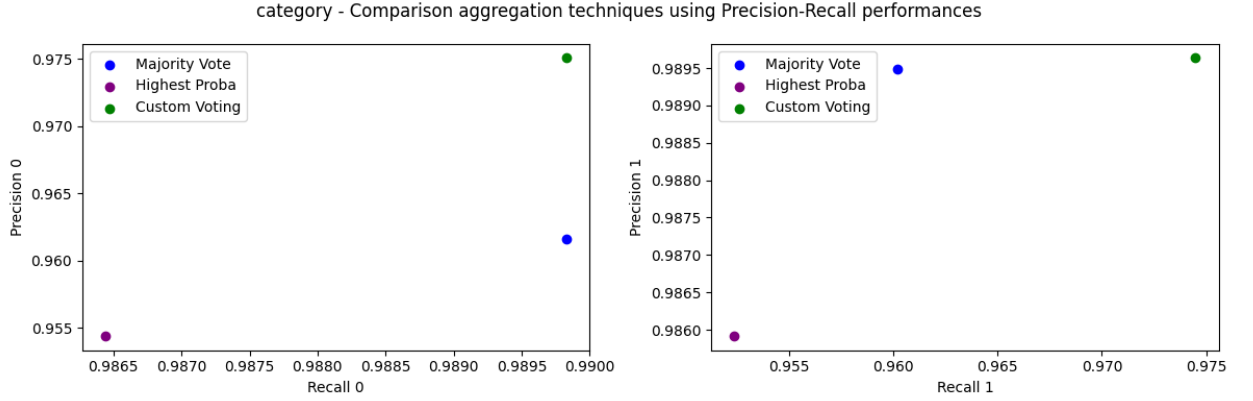


Figure 4.12: Aggregation comparison performances for unit anomalies

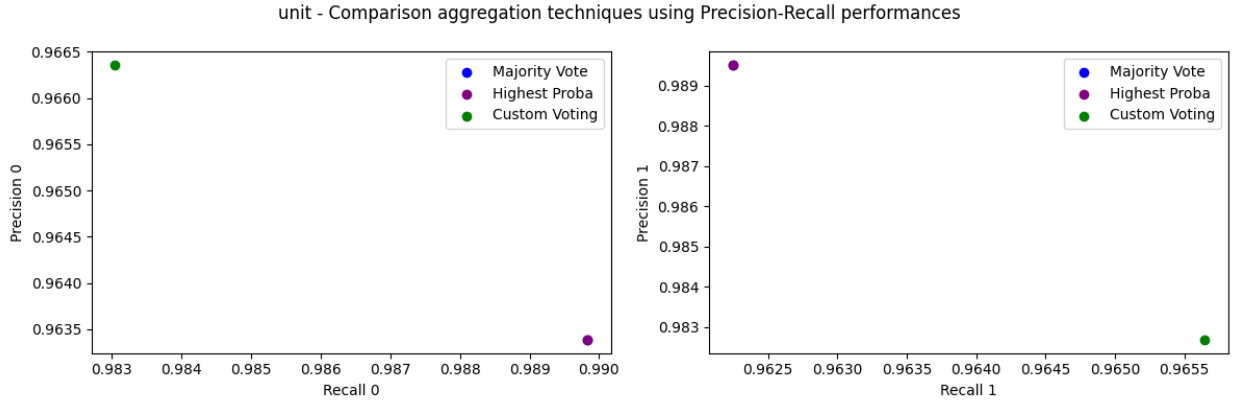


Figure 4.13: Aggregation comparison performances for categories anomalies

It is pretty clear that each method has great performances but for most of the cases, Custom Voting renders a better recall for the anomalies and a better precision for the correct values. This was expected since our weights favour more models that follow those performances during training combined with a high confidence in their prediction. Majority Vote relies too much on the quality of the precision metrics, which is not what we are looking for. Hence, this method will not be chosen. However, we observe that for driver, the individual qualities of models provide better scores through the Highest Probability technique. The result is close to Custom Voting output, but giving its better recall, we decide to keep this aggregation technique for driver anomalies.

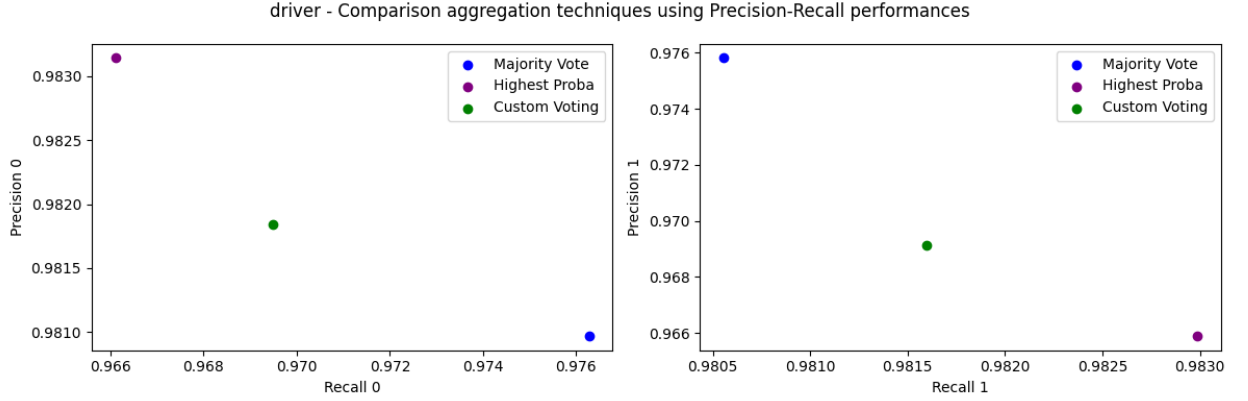


Figure 4.14: Aggregation comparison performances for driver anomalies

4.2.2 Models contribution

Working with ensemble learning adds robustness to our final output, however this works only if one or some models do not outperform and crush all other classifier.

The highest probability aggregation technique takes the output of the most confident model. It allows us to verify the contribution of each model by counting the number of times each of their output is selected. We compute the mean of each model's contribution on 10 runs with different test set for each type of anomaly.

Model ID	0	1	2	3	4	5	6	7	8	9
Contribution	98	57	91	73	45	44	43	49	44	41

Table 4.2: Mean of model's contribution for Specific Scaling anomalies

Model ID	0	1	2	3	4	5	6	7	8	9
Contribution	38	27	44	50	120	52	121	54	35	43

Table 4.3: Mean of model's contribution for Categories anomalies

We clearly note that some model are stronger than others, especially for scaling. However, there are no model that do not contribute at all, meaning that they may all impact the output for the **Majority Vote** or the **Custom Voting**. Models trained for unit and driver follow the same behaviour.

4.3 Testing and comparison

4.3.1 Clustering algorithm

Anomaly detection tools usually rely on unsupervised algorithms, especially their ability to create clusters that differentiate the correct values from the errors. In this perspective, DBSCAN separates normal data from noisy data that reflects the anomalies. This algorithm was explained in the chapter 1. We used this technique on our test set to compare it to our ADTK. However, the results displayed in figure 4.15 are not convincing compared to our architecture.

	Scaling	Unit	Category	Driver
Accuracy	0.763158	0.511545	0.5247	0.516808

Figure 4.15: DBSCAN performances on anomaly type

The maximum performance reached is around 75% for scaling anomalies using $eps = 10$ and $min_samples = 20$ as parameters. This algorithm struggles a lot to find the noisy data for all other type of anomaly, reaching maximal accuracy of 55%. This is due to the difficulty of creating clusters with our versatile data in high dimension. Strictly speaking, DBSCAN creates one cluster with correct values and considers all points that do not belong to it as anomalies. Therefore, the results show us that too much anomalies are considered as correct values due to their proximity to other points.

4.3.2 Generalization

To certify the good generalization of our specialized models, we have removed from the training set all reports (thus all ADs) of a specific company and added them in the test set. We can observe that there are no major differences between both accuracies: this can be interpreted as a good generalization by our models. Even when they've never heard of a company, they are able to properly find anomalies

4.3.3 One error per line

The training of our model has been conducted with a line containing a single wrong value. Therefore, our ADTK is created to detect an error if and only if all other values of the record are correct. This could be a huge limitation regarding the

number of possible errors in a single line.

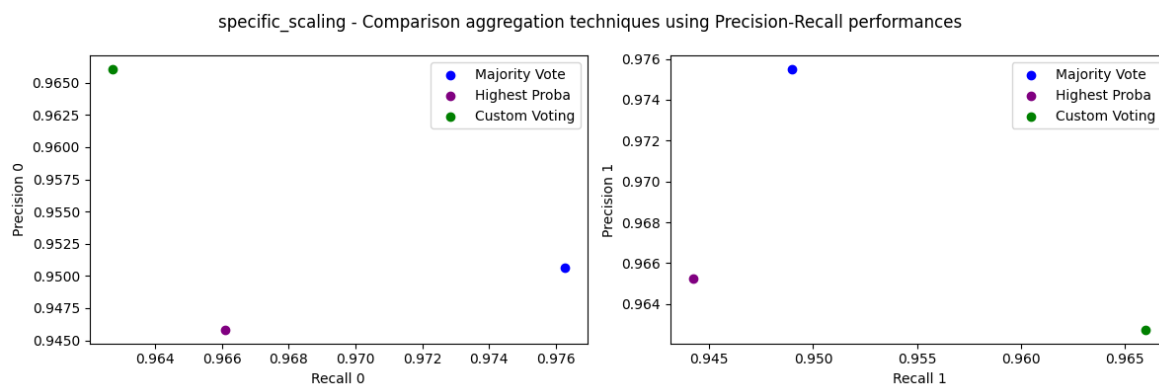


Figure 4.16: Performances comparison on test records containing scaling and category anomalies

However, depending on the feature importance, we may have multiple errors on the same line that are detected by different ensemble blocks. Figure 4.16 represents a test conducted with all anomalous records containing at least two anomalies. We observe that our models are performing well even when 2 values are wrong in the same line. The scaling *ensemble block* is able to get 96% accuracy with custom voting while all anomalies records have an error in the activity data value and category. This highlight the fact that our model relies on many features and that this possible limitation can be handled by the tool.

4.4 Test with real carbon report

Test on real unseen data collection were conducted thanks to the collaboration of Tapio's experts. We were able to try out the ADTK directly on carbon reports, filled by two different companies, that were not yet reviewed by human.

In the first report on figure 4.17, 5 lines were classified as scaling anomalies. As a reminder, this error focuses on badly scaled activity data value.

We easily observe that for lines 1 and 2, the value was wrongly encoded due to a confusion between the k€ and € unit. The last line has also a huge value which does not seem coherent regarding the unit and the category of emission. Those 3 records were confirmed to be anomalies by Tapio's experts.

	nb of employee	total_emission	activity data category	activity data scope	activity data value	activity data unit	emission factor value	emission factor unit	driver name
0	167.3	1.949790e+08	Capital Goods	Scope 3	212627.000	k€	917.000	k€	Capital goods
1	167.3	4.784216e+08	Business Travel	Scope 3	402035.000	k€	1190.000	kgCO2e/k€	Mobility
2	167.3	2.590000e+00	Upstream Leased Assets	Scope 3	72.000	kWh	0.036	kgCO2e/kWh	Capital goods
3	167.3	2.900000e-01	Upstream Leased Assets	Scope 3	8.000	kWh	0.036	kgCO2e/kWh	Capital goods
4	167.3	6.275241e+06	Upstream Leased Assets	Scope 3	2510096.496	m³	2.500	kgCO2e/m³	Energy Consumption

Figure 4.17: ADTK detection on real carbon report 1

The lines 3 and 4 has been detected by the ADTK because the value are low. However, the error was in the unit which should have been in "unit" and not in "kWh". Since the error could have been located in the value, it is good that the ADTK detected it, even as a scaling error. But the unit *ensemble block* should have detect those 2 lines. This was not the case, mainly because "kWh" is a justified choice as unit for the given line regarding the other attributes such as the category or driver. This highlight the link between the unit and scaling *ensemble blocks* and lead us to reconsider how we trained them. A more specific border should be found to differentiate them during the training.

In the second report on figure 4.18, 3 types of anomaly were detected. The attribute classified as anomaly is highlighted in yellow for each row.

	nb of employee	total_emission	activity data category	activity data scope	activity data value	activity data unit	emission factor value	emission factor unit	driver name
0	45.000000	201500.000000	Upstream Leased Assets	Scope 3	310.000000	m² net floor area	650.000000	kgCO2e/m² net floor area	Capital goods
1	45.000000	720.550000	Upstream Leased Assets	Scope 3	0.981350	k€	734.240000	kgCO2e/k€	Energy Consumption
2	45.000000	87.900000	Upstream Leased Assets	Scope 3	1.000000	unit	87.900000	kgCO2e/unit	Capital goods

Figure 4.18: ADTK detection on real carbon report 2

In the first row, a wrong value was detected in a "capital good lead asset" activity data. According to carbon experts, the value itself was correct but it has not been amortized using the lifetime. Indeed, the value should be "divided" into smaller values to fit with the number of year on which you amortize its emissions.

The second line is an incorrect detection, this record's driver is good. We thus have an example of the ADTK with a high recall on anomalies. It is not surprising either since ADs categorized as "Upstream Leased Assets" are often classified in the "Capital Goods" driver.

The last line was detected as a category error which has been confirmed to be an effective anomaly.

According to Tapio's experts, no other predictable anomalies has been found in this carbon report. This highlights the recall quality of the ADTK.

Chapter 5

Discussion

5.1 Thesis process with Tapio

5.1.1 Why Tapio

Doing a thesis in a company has many advantages. First of all, the thesis has entirely been developed in a realistic environment as the project is based on real world data and tackles Tapio's issues. Moreover, integrating a research work in a running company yields different skills that cannot be brought by a classical academic approach such as adaptability, communication with other team of employee, constraint management. In addition, there is a need of being able to deal with not traditional issues that comes within the scope of a master thesis. Finally, researching in a domain that makes sense and has a real impact on global problem is motivating.

5.1.2 Tapio's goal

It is important to highlight the fact that the purpose of this thesis was not to create a fully operational tool that would directly be integrated in Tapio's process. The aim was mainly to focus on discovering if AI, more specifically machine learning, could help in the new domain of climatic data. Anomalies detection is one of the numerous application of machine learning that can ease the process to reach global carbon neutrality. Getting to know the long process to elaborate such architecture to answer a problem is a rich knowledge for small companies like Tapio. Hence, the whole machine learning models creation procedure has been followed by Tapio's members and will be reused for future enhancement.

5.2 Integration in Tapio's process

5.2.1 When to use the ADTK

The presented ADTK has mainly been conceptualised to detect outrageous errors that will make Tapio's expert lose time because of their potential amount. Therefore, this tool should mostly be used after the first data collection from the client itself. This gathering of data is the first step to create a carbon report and the carbon experts has not yet been requested at this stage. Tapio's data collect tool is displayed in figure 5.1 where you can see that each line represents an activity data, grouped by driver. Completing this data is quite an easy process but may be a long labour depending on the company. Hence, it can lead to some basic mistakes like the one our ADTK is trained to detect.

The screenshot shows a web application titled "CAPITAL GOODS - FROM SUPPLIERS". It contains two main sections for data entry:

- IT & ELECTRONIC MATERIAL**: This section has a table with the following data:

Description	Méthode	EF	Valeur	Incertitude	Émissions totales
Ordinateurs	Qua...	Computer...	10	Données brutes - 5...	1560 kgCO ₂ e
- VEHICLES**: This section has a table with the following data:

Description	Méthode	EF	Valeur	Incertitude	Émissions totales
Description	Auc...	Sélectionner f...	Valeur	Données brutes - 5...	0 kgCO ₂ e

Figure 5.1: Data collection process example

However, as mentioned in previous chapters, the ADTK would also help experts in their correction to keep coherence between each other. We could thus run the tool again after Tapio's member have made their changes to the data collection. This will make sure the expert has not forgotten anomalies and that he follows the same logic than other Tapio's experts.

5.2.2 Reporting

As showed in figure 1.3 in chapter 1, when introducing unseen data, our Anomalies Detection ToolKit will broadcast those records to each *ensemble block* and if one of them detects an anomaly, it is reported directly. However, telling only that an anomaly is present is not enough. Fortunately, the fact that our models are specialized allow us to give more insights on the type of anomaly they found. By using the ADTK, the carbon expert will see warnings on each records that our tool

labelled as anomaly. This warning will give basic information such as the type of anomaly and the probability associated to it. To illustrate the reporting process, we take as example the figure 5.1 and complete the second line with an incorrect value: 4000 cars for a small company. The result is shown in figure 5.2. The ADTK would display an overview of all ADs, including a little warning on lines containing errors with a specific messages to help the user as shown in figure 5.3. The message could appear when hovering on the warning emote.

The screenshot shows the ADTK interface for 'CAPITAL GOODS - FROM SUPPLIERS'. It has two main sections: 'IT & ELECTRONIC MATERIAL' and 'VEHICLES'. Each section has a table with columns: Description, Method, EF, Value, Uncertainty, and Total emissions. In the 'VEHICLES' section, the entry for 'Voiture' has a value of 4000, which is highlighted in red, indicating an anomaly. The 'Total emissions' for this entry is 25648000 kgCO₂e.

Figure 5.2: Data collection scaling anomaly example

DESCRIPTION	EMISSION FAC...	VALUE (UNIT)	TOTAL EMISSI...	MAIN STEP	DRIVER	CATEGORY	ACTIONS
Voiture	Small car (Segmer	4000 unit	25648 t CO ₂ e	Capital goods	Capital goods	Capital Goods	
Ordinateurs	Computer - Laptop	10 unit	1.56 t CO ₂ e	Capital goods	Capital goods	Capital Goods	

Figure 5.3: Data overview scaling anomaly warning

We may imagine many ways to generate those warnings. As a first version, the experts could trigger a button related to a carbon report when an anomaly check is needed. This gives the full control of the tool to the carbon experts during the first period of use of the ADTK. It would also save calculation time if the experts consider the use of the tool useless for some specific carbon reports. In a more advanced version, the tool could be triggered for each entry of a new activity data in Tapio's platform. When a client fills its carbon report, each AD would be directly flagged by the ADTK as a correct record or an anomaly. Hence, the client would be totally autonomous and could correct instantly its potential errors.

5.3 Limitations

5.3.1 Specific anomalies

An important limitation of our tool is the type of anomalies it can detect. Indeed, model are trained to detect specific anomalies pre-determined for which Tapio's experts have validated the creation process. There are a numerous errors in a client's carbon report for which it would be very difficult to find patterns or rules to recreate them automatically. Hence, it is important to note that the tool is made to detect only identified anomalies and that it does not cover all errors that we could find in a carbon report. This makes sense since the aim of the tool is to provide feedback to carbon experts on foolish but hard to find errors. The ADTK is mainly made to save time to experts. However, we could imagine a dedicated database in which experts could flag more difficult errors encountered. This would give us a full training set to use for creating new specialized models made for odd anomalies detection.

5.3.2 Limited to Tapio's data

It is important to clarify that our ADTK is Tapio's data dependent. Obviously, the nature of machine learning modelling pushes towards the best generalization to not overfit the data on which the training is done. However, it would be a tough task to apply our tool on data coming from another source. Firstly, it would require to build a matching algorithm to convert the data into the same state as Tapio's data. Indeed, the sets need to undergo the preprocessing and thus need to have the exact same shape and types as the training set. Secondly, since climate data are highly dependent on how the user complete the data, the ADTK could not work as well as with Tapio's data.

5.4 Improvements

5.4.1 Flexibility

We have already stated the flexibility of our architecture. We allow to choose any performing machine learning algorithm for given models of a single *ensemble block*. We also manage to keep different aggregation technique depending on their efficiency and the goal for a type of anomaly. Even the alpha beta value in the custom voting technique are different for all types of anomaly. However, we could go much deeper into the freedom given to our choices by, for example, shuffle different type of algorithm into a single *ensemble block*. Regarding the high performances of other algorithm such as Random Forests, it could help generalization to use

different algorithms because they will not calculate their output in the same way. With a lot more data, we could also try to introduce some Neural Networks in the pool of algorithm that we could use. This would require many experiences and testing.

5.4.2 Meta-model for aggregation

We currently proposed 3 distinct aggregation techniques, but many other exist. An interesting one regarding our architecture could be what we generally call a **Meta-model**. This kind of model is used for example in stacked based ensemble machine learning such as explained in [Buyrukoğlu and Savaş, 2023]. In a nutshell, this model would be the aggregation itself and would be trained using the outputs of all models of a specific *ensemble block* in addition of the training data. This model could also have its hyper-parameter fine tuned to favour the performances of `recall_1` and `precision_0`. To use such model, we would need to have much more data in order to train each of the *ensemble block* model with significantly different sets. Hence, it is not possible to implement such technique regarding the amount of data available but could be a nice to have feature in no time.

5.4.3 Dispenser’s pipeline

The current ADTK is trained using only Tapio’s data. As explained previously, it mainly limits its use for data collected by Tapio. In addition, one of the goal of the ADTK was to bring coherence between Tapio’s experts. However, Tapio’s tool could be use by other people such as dispensers ¹ that would follow other rules to deal with the data. We could thus imagine a process to retrieve data of specific dispensers to train others ADTK and thus create single pipelines that are personalized for each external user of the platform. Therefore, each dispenser could have their own Pipeline that are aligned with their way of interpreting and managing the data collected.

5.4.4 Proposition

For now, our ADTK is only a **detection** tool that flags anomalies that it finds. However, with specific type of anomalies, we could imagine a correction or a proposition of the wrong value detected. For example, this could be done for the driver anomalies because the number of different output possibilities is limited to 7 values. Hence, an approach would be to train our models with 7 classes instead of 2. This is called multi-label classification. It would allow us to firstly detect

¹Dispensers are external carbon expert that would use Tapio’s platform to manage their clients

a driver anomaly in a record and secondly propose the value that is most likely correct. This could be extend to the category anomalies after further testing.

However, for continuous values such as activity data value, the approach to do proposition would be completely different. We would need to train our models to do regression and not classification. Hence, our tool could create some ranges in which the value should be included but it would be hard to propose a **exact** correction of the error.

5.4.5 Textual anomalies detection

To detect anomalies on textual data such as category or driver, we use encoding to convert them into numerical values. However, for features such as activity data (AD) name or emission factor (EF) name, the amount of possibilities is huge and it would not be efficient to encode them. In order to deal with those kind of value, we would need to use Natural Language Processing (NLP) ² to detect the main subjects of a given input and check whether it is coherent regarding other values.

This would be especially interesting for the EF selection because it is a tough and long task to find the EF corresponding to the given AD. Therefore, it is a great source of mistakes. We would thus need to verify if the chosen EF is coherent with respect to the name of the encoded AD. This would require much research in the NLP field which is a complex machine learning domain. In addition, we would need much more data to be able to train models to make good recognition and to verify the similarity between EF name and AD description.

²Natural language processing (NLP) refers to the branch of artificial intelligence with giving computers the ability to understand text and spoken words in much the same way human beings can

5.5 Conclusion

We stated that a valuable number of anomalies can be found in the process of data collect for a carbon report creation. In the scope of this thesis, anomalies are considered to be strictly incorrect values. Moreover, the data verification and correction is long and mobilizes a full time carbon expert. Lastly, it is difficult if not impossible to establish by hand rules or threshold to find automatically such anomalies. Therefore, we propose an Anomalies Detection ToolKit that will use machine learning models to flag rows containing anomalies and deliver enough information to understand quickly where the error is.

The data is a core aspect of this work. We describe the data used and detail all modifications or enhancements that were made to upgrade its quality. This involves many data preparation techniques that are able to make our data as good as possible for training. We particularly focus on the fact that Tapio’s data is semi-supervised and contains only correct values. Hence, we propose a precise process using bootstrap learning to generate coherent records that are define as anomalies. This allow us to treat this issue as a supervised problem in machine learning.

The ADTK includes multiple *ensemble blocks* that are specialized to detect a specific type of anomaly. Each of them is composed of a fixed number of machine learning models that are trained with a unique data set containing correct records and anomalies of a specific type. When using the ADTK with new data, each *ensemble block* will aggregate the outputs of all its models to tell if the given input contains anomalies. This aggregation uses a pre-determined technique among 3: Majority Vote, Highest Probability and Custom Voting.

The tests conducted demonstrate that we are able to obtain tremendous results on each type of anomaly. The metrics used to determine the performances of our ADTK are recall, precision and accuracy. A strength of the proposed architecture is its flexibility, hence the fact that we may use any algorithm for any model and any aggregation technique in any *ensemble block*. XGBoost is the best algorithm for scaling, unit and driver anomalies detection while Random Forest performs the best for category anomalies detection. We use the aggregation technique that favour the most the recall of anomalies since we would rather wrongly flag a record than miss an error. Therefore, we aggregate using Custom Voting for scaling, unit and category and using Highest probability for driver anomalies.

We were able to run tests using real data collected from clients after the beginning of the testing phase. The ADTK came out well by flagging multiple records as

anomalies that were confirmed to be effective errors in the data collected. It also made a few mistakes but didn't miss out any anomalies that it should have detected.

To conclude, this work demonstrates that we were able to compose and test a complete architecture that detects pre-determined anomalies using highly versatile and limited data. Many known techniques allow to create anomalies detection tools using unsupervised learning, such as clustering. However, this approach was not efficient regarding the data we process. As previously stated, the use of machine learning to solve the issues was mandatory since it is impossible to establish a rule-based solution. We were able to combine a few existing and validated methods to reach the goal we fixed. Bootstrap sampling was necessary to create anomalies, Decision trees are powerful to deal with binary classification and ensemble learning help us having a robust and flexible solution.

Bibliography

- [Abdallah et al., 2017] Abdallah, Z. S., Du, L., and Webb, G. I. (2017). Data preparation.
- [Abe, 2001] Abe, S. (2001). *Generation of Training and Test Data Sets*, pages 239–247. Springer London, London.
- [Ahsan et al., 2021] Ahsan, M. M., Mahmud, M. A. P., Saha, P. K., Gupta, K. D., and Siddique, Z. (2021). Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9(3).
- [Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).
- [Brownlee, 2020] Brownlee, J. (2020). *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery.
- [Buyrukoğlu and Savaş, 2023] Buyrukoğlu, S. and Savaş, S. (2023). Stacked-based ensemble machine learning model for positioning footballer. *Arabian Journal for Science and Engineering*, 48.
- [Chalapathy and Chawla, 2019] Chalapathy, R. and Chawla, S. (2019). Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407.
- [Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(15):1–58.
- [Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16.
- [Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.
- [Dixon, 2006] Dixon, P. M. (2006). *Bootstrap Resampling*. John Wiley Sons, Ltd.

- [Douiba et al., 2022] Douiba, M., Benkirane, S., Guezzaz, A., and Azrour, M. (2022). Anomaly detection model based on gradient boosting and decision tree for iot environments security. *Journal of Reliable Intelligent Environments*.
- [Duboue, 2022] Duboue, P. (2022). *Feature Engineering*, pages 109–127. Springer International Publishing, Cham.
- [Fernández et al., 2018] Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from Imbalanced Data Sets*. Springer Cham.
- [Goernitz et al., 2013] Goernitz, N., Kloft, M., Rieck, K., and Brefeld, U. (2013). Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262.
- [Gujral, 2023] Gujral, E. (2023). Survey: Anomaly Detection Methods.
- [Han et al., 2020] Han, Y., Jin, Z., and Liu, Y. (2020). Sentiment analysis via semi-supervised learning: a model based on dynamic threshold and multi-classifiers. *Neural Computing and Applications*, 32:5117–5129.
- [Hapke and Nelson, 2020] Hapke, H. and Nelson, C. (2020). *Building Machine Learning Pipelines*. O’Reilly Media.
- [Hawkins, 1980] Hawkins, D. M. (1980). *Identification of Outliers*. Springer Dordrecht.
- [Ikram et al., 2021] Ikram, S., Cherukuri, A. K., Poorva, B., Ushasree, P., Zhang, Y., Liu, X., and Li, G. (2021). Anomaly detection using xgboost ensemble of deep neural network models. *Cybernetics and Information Technologies*, 21:175–188.
- [Jalal and Ezzedine, 2020] Jalal, D. and Ezzedine, T. (2020). Decision tree and support vector machine for anomaly detection in water distribution networks. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1320–1323.
- [Jijo and Mohsin Abdulazeez, 2021] Jijo, B. and Mohsin Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2:20–28.
- [Jiménez et al., 2007] Jiménez, Á. B., Lázaro, J. L., and Dorronsoro, J. R. (2007). *Finding Optimal Model Parameters by Discrete Grid Search*, pages 120–127. Springer Berlin Heidelberg, Berlin, Heidelberg.

- [Keogh and Mueen, 2017] Keogh, E. and Mueen, A. (2017). *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA.
- [Kulesa et al., 2015] Kulesa, A., Krzywinski, M., Blainey, P., and Altman, N. (2015). Sampling distributions and the bootstrap. 12(6):477–478.
- [Lazarevic and Kumar, 2005] Lazarevic, A. and Kumar, V. (2005). Feature bagging for outlier detection. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 21–24, Chicago, Illinois, USA.
- [LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521.
- [Li et al., 2020] Li, M., Fu, X., and Li, D. (2020). Diabetes prediction based on xgboost algorithm. *IOP Conference Series: Materials Science and Engineering*, 768(7):072093.
- [Littlestone and Warmuth, 1994] Littlestone, N. and Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation*, 108(2):212–261.
- [Melo, 2013] Melo, F. (2013). *Receiver Operating Characteristic (ROC) Curve*, pages 1818–1823. Springer New York, New York, NY.
- [Nguyen et al., 2010] Nguyen, H. V., Ang, H. H., and Gopalkrishnan, V. (2010). Mining outliers with ensemble of heterogeneous detectors on random subspaces. In Kitagawa, H., Ishikawa, Y., Li, Q., and Watanabe, C., editors, *Database Systems for Advanced Applications*, pages 368–383. Springer and Berlin and Heidelberg.
- [Omar et al., 2013] Omar, S., Ngadi, A., and Jebur, H. H. (2013). Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, 79(2).
- [Ramaswamy et al., 2000] Ramaswamy, S., Rastogi, R., and Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438.
- [Refaeilzadeh et al., 2009] Refaeilzadeh, P., Tang, L., and Liu, H. (2009). *Cross-Validation*, pages 532–538. Springer US, Boston, MA.
- [Schiefer and Schiefer, 2021] Schiefer, H. and Schiefer, F. (2021). *Correlation*, pages 95–98. Springer Fachmedien Wiesbaden, Wiesbaden.

- [Sessions and Valtorta, 2006] Sessions, V. and Valtorta, M. (2006). The effects of data quality on machine learning algorithms. pages 485–498.
- [Ting, 2010] Ting, K. M. (2010). *Precision and Recall*, pages 781–781. Springer US, Boston, MA.
- [Tlameo et al., 2021] Tlameo, E., Thabiso, M., Dimanev, M., Thabo, S., Banyatsang, M., and Oteng, T. (2021). A survey on missing data in machine learning. *Journal of Big Data*, 8.
- [Webb, 2010] Webb, G. I. (2010). *Overfitting*, pages 744–744. Springer US, Boston, MA.
- [Xu et al., 2019] Xu, X., Liu, H., and Yao, M. (2019). Recent progress of anomaly detection. *Complexity*, 2019:11.
- [Zhang and Ma, 2012] Zhang, C. and Ma, Y. (2012). *Ensemble Machine Learning*. Springer.
- [Zhang et al., 2018] Zhang, D., Jiao, L., Bai, X., Wang, S., and Hou, B. (2018). A robust semi-supervised svm via ensemble learning. *Applied Soft Computing*, 65:632–643.
- [Çelik et al., 2011] Çelik, M., Dadaşer-Çelik, F., and Şakir Dokuz, A. (2011). Anomaly detection in temperature data using dbscan algorithm. In *2011 International Symposium on Innovations in Intelligent Systems and Applications*, Istanbul, Turkey. IEEE.

Appendices

Appendix A

Tapio

A.1 Feature description of training set

Table A.1: Tabular of original data features selected to train our models

Feature	Description	Type
Company	Name of the company	String
Sector	Sector in which the company operates (see A.2)	String
Report Name	Name of the Report that contains the AD	String
Employee	Number of employee of the company at this moment	Integer
Date	Date of the report	datetime
Total emission	Total calculated emissions of the AD (AD.value x EF.value)	Float
AD category	Category to which the AD belongs (see 1.1)	Integer
AD driver	Driver to which the AD belongs (see ??)	Integer
AD scope	Scope to which the AD belongs (see 1.1)	Integer
AD value	Value of the AD in the specified unit	Float
AD unit	Unit of the AD	String
AD uncertainty	Uncertainty on the AD value calculation (see A.2)	Float
EF value	Value of the specific EF in the specified unit	Float
EF unit	Unit of the EF, usually $\frac{tCO_2eq}{ADunit}$	String
EF uncertainty	Uncertainty on the EF value calculation (see A.3)	Float

A.2 Tapio's driver

N°	DRIVER	SECTIONS	SOURCES OF EMISSIONS
1	Energy consumption	On-site fuel consumption On-site electricity consumption On-site cooling system	Fuel Green electricity Grey electricity Refrigerants
2	Mobility	Company owned vehicles Employee commuting Business travels	Combustion vehicles Electric Vehicles with grey electricity Electric vehicles with green electricity Vehicles not owned by the organisation Public transport Business travels by vehicles not owned by the organisation Business travels by public transport Business travel by plane
3	Logistics	Company owned vehicles Upstream logistics Downstream logistics	Combustion vehicles Electric vehicles grey electricity Electric vehicles green electricity Upstream logistics Downstream logistics
4	Goods and services	Purchased goods and services	Metals Plastics Glass Paper/Cardboard Construction material Textiles Chemicals Food Data consumption Other Non material services Water
5	Waste	Solid waste Wastewater	Solid wastes Wastewater
6	Capital goods	Capital goods Assets leasing Leased assets usage Franchises Financial investments	For Capital goods and assets leasing: Construction Furniture and other manufactured goods IT & electronic material Vehicles Tools and machinery Textiles Fuel Electricity Refrigerants Other Financial investments
7	Sold products	Process of sold products Use of sold products	Fuel Electricity Refrigerants Data consumption Other

Figure A.1: List of drivers created by Tapio [Tapio 2019]

A.3 Tapio's sectors

Sectors
Alimentation
Cities
Communication
Events
Fashion & Textiles
Others
Products
Services
Tech & Digital

Table A.2: List of sectors used by Tapio

A.4 Tapio's calculation of uncertainty

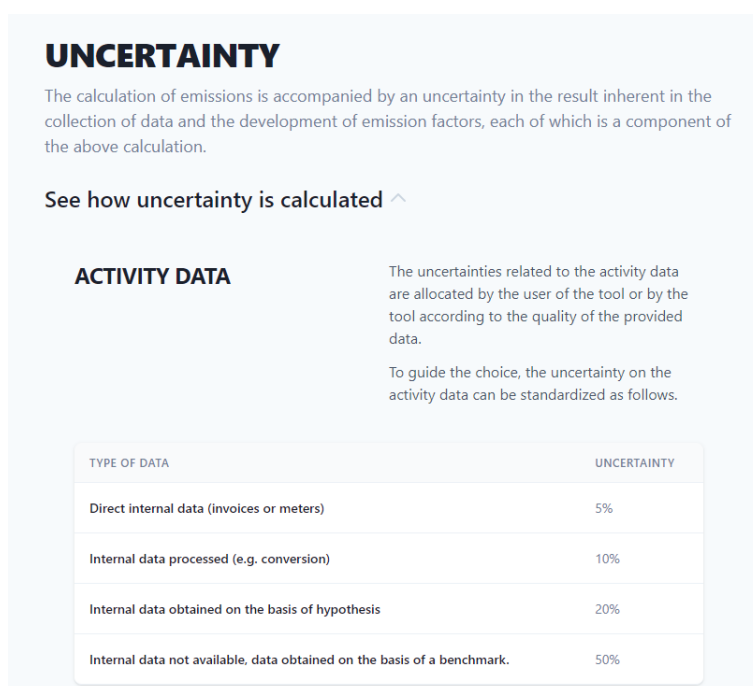


Figure A.2: Calculation of uncertainty of AD value [Tapio 2019]

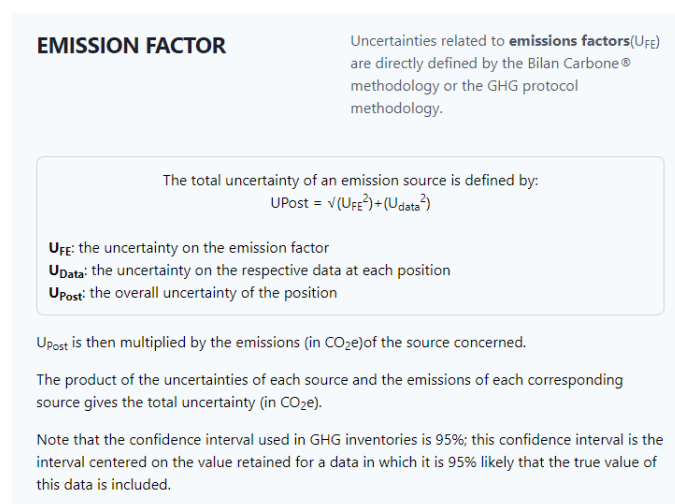


Figure A.3: Calculation of uncertainty of EF value [Tapio 2019]

Appendix B

Algorithm details

B.1 RandomForest

RandomForests is a Decision Tree based algorithm that uses **bagging**. Bagging creates a different training subset from sample training data with replacement the final output is based on an aggregation technique (averaging or majority vote).

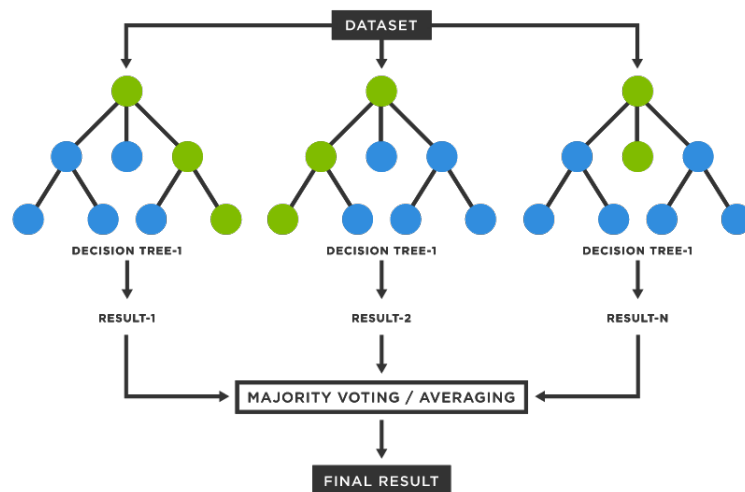


Figure B.1: Random Forest process (TIBCO ¹)

B.2 XGBoost

XGBoost is a Decision Tree based algorithm that uses **boosting**. Boosting is an iterative algorithm that will create a tree and its output given the training data. The next tree will then be created using the output of the first tree. More precisely, it gives weight on wrongly classified sample at each iteration to optimize a determined function (here the gradient). It is a strong algorithm for anomaly detection because if an anomaly is not detected, the next iteration will put a higher weight on this specific record and will have more chance to be label it as anomaly.

XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions)

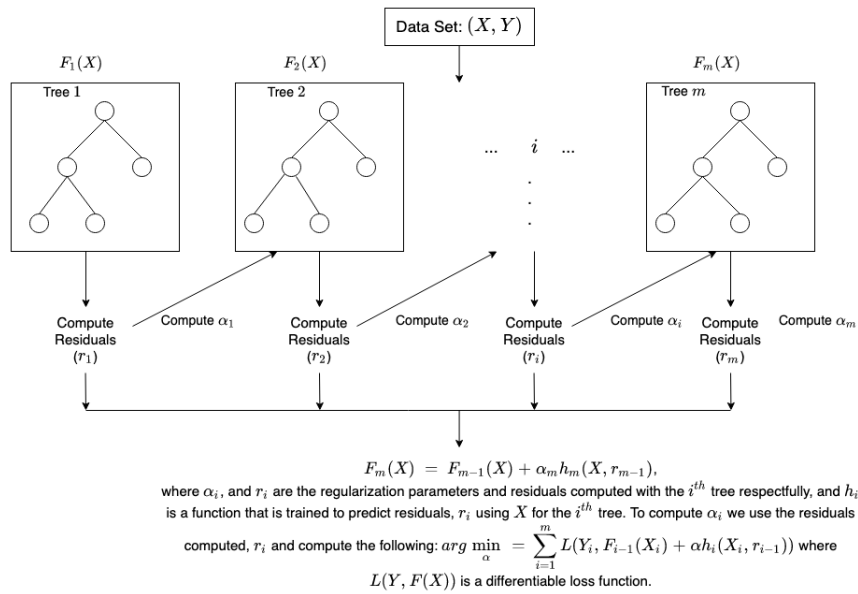


Figure B.2: XGBoost process (AWS documentation ²)

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl