UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Automated classification of service reports using natural language processing techniques

End of Degree Project

Bachelor's Degree in Data Science

AUTHOR: Gilyarovskaya , Elizaveta Alexeevna

Tutor: Morillas Gómez, Samuel

External cotutor: BERG, KARE OLAV

ACADEMIC YEAR: 2021/2022

Acknowledgements

I would like to express my deepest gratitude to my advisor and mentor Jan Erik Hjelseth who offered guidance and support throughout my whole internship and made this project possible. His experience and approach to research and data science is a source of inspiration for me. Jan Erik Ofsti, Tore Tveit, Steve May are amazing team and I truly appreciate having the opportunity to work with such a professionals.

Dear Ignacio Navarro Cano, Álvaro Mazcuñán, Miquel Marín and Ángel Langdon thank you all for being amazing teammates in all my projects and deliverables, you are the ones I have learned the most from and I would not be where I am without you all.

Last but not least, I wish to extend my special thanks to my supervisor Samuel Morillas who has offered himself as tutor even though he was on vacation, thank you for your invaluable feedback.

Elizaveta Gilyarovskaya

Resum

Kongsberg Maritime és una empresa tecnològica que lliura sistemes de posicionament, topografia, navegació i automatització a vaixells mercants i instal·lacions a alta mar. És de vital importància tenir un sistema madur i eficaç de gestió de reports que permeti analitzar i classificar tota la informació disponible i recolzar així el procés de la presa de decisions. L'empresa es beneficiarà de la proposta desenvolupada en aquest projecte de forma directa ja que permet automatitzar el procés de revisar i extreure informació dels informes i dedicar els recursos antigament implicats en això a altres àrees del negoci

Com a part de *Equip d'Innovació de Dades* he desenvolupat aquest projecte amb l'objectiu d'automatitzar els processos interns de la companyia aplicant tècniques d'intel·ligència artificial, més concretament de processament del llenguatge natural, a les tasques de revisió i classificació dels informes de manteniment realitzats pels enginyers de servei. En primer lloc, s'ha desenvolupat un model d'aprenentatge semisupervisat, *few-shot learning*, per a la tasca de classificar el text d'un informe a les categories d'interès. Quan les prediccions d'aquest model han estat corregides per experts del domini i s'han generat suficients dades etiquetades, s'han entrenat classificadors supervisats per millorar les prediccions i arribar a un model definitiu per posar-lo en producció. També es va entrenar un model *deep learning* de tipus red transformers mitjançant la tècnica *transfer learning*. Finalment, s'ha dissenyat *custom rule-based matching* per al reconeixement i l'extracció d'entitats i paraules clau d'interès.

Per posar el programa en producció, s'ha desenvolupat una interfície d'usuari que mostra la sortida dels models de forma gràfica i permet als agents encarregats de supervisar-lo introduir les correccions a les prediccions del model per a ajustar-les i millorar-les posteriorment.

Paraules clau: Processament de llenguatge natural, aprenentatge automàtic, classificació de text no supervisada, classificació de text supervisada, extracció d'entitats personalitzades, automatització, millora empresarial, informes d'enginyeria de serveis

Resumen

Kongsberg Maritime es una empresa tecnológica que entrega sistemas de posicionamiento, topografía, navegación y automatización a buques mercantes e instalaciones en alta mar. Es de vital importancia para ellos tener un sistema maduro y eficaz de gestión de reportes que permita analizar y clasificar toda la información disponible y apoyar de esta forma el proceso de la toma de decisiones. La empresa se beneficiará de la propuesta desarollada en este proyecto de forma directa ya que permite automatizar el proceso de revisar y extraer información de los informes y dedicar los recurosos antiguamente implicados en eso a otras áreas del negocio.

Como parte del *Equipo de Innovación de Datos* he desarrollado este proyecto con el objetivo de automatizar los procesos internos de la compañía aplicando técnicas de inteligencia artificial, más concretamente de procesamiento del lenguaje natural, a las tareas de revisión y clasificación de los informes de mantenimiento realizados por los ingenieros de servicio. En primer lugar, se ha desarrollado un modelo de aprendizaje semisupervisado, *few-shot learning*, para la tarea de clasificar el texto de un informe en las categorías de interés. Una vez que las predicciones de este modelo han sido corregidas por expertos del dominio y se han generado suficientes datos etiquetados, se han entrenado clasificadores supervisados para mejorar las predicciones y llegar a un modelo definitivo para ponerlo en producción. También se entrenó un modelo *deep learning* de tipo red transformers mediante la técnica *transfer learning*. Por último, se ha diseñado *custom rule-based matching* para el reconocimiento y extracción de entidades y palabras clave de interés.

Para poner la herramienta en producción, se ha desarrollado una interfaz de usuario que muestra la salida de los modelos de forma gráfica y permite a los agentes encargados de supervisarlo introducir las correcciones a las predicciones del modelo para su posterior ajuste y mejora.

Palabras clave: Procesamiento de lenguaje natural, aprendizaje automático, clasificación de texto no supervisada, clasificación de texto supervisada, extracción de entidades personalizadas, automatización, mejora empresarial, informes de ingeniería de servicios

Abstract

Kongsberg Maritime is a technology enterprise that delivers systems for positioning, surveying, navigation, and automation to merchant vessels and offshore installations. It is of critical importance for them to have a mature and effective reporting management system that allows analyzing and classifying all the available information to support the decision-making process. The company will benefit directly from this proposal since it will allow to automate the process of reviewing and extracting information from the reports and dedicate the resources formerly involved in that to other areas of the business.

As part of the *Data Innovation Team* I have developed this project with the aim of automating the company's internal processes by applying artificial intelligence, more specifically, natural language processing techniques to the tasks of reviewing and classifying maintenance reports made by service engineers. First, a semi-supervised learning model, *few-shot learning*, has been developed for the task of classifying the text of a report into the categories of interest. Once the predictions of this model have been corrected by domain experts and enough labeled data has been generated, supervised classifiers were then trained to improve the predictions and come up with a definitive model to put into production. A *deep learning* transformers type of model was also trained using *transfer learning* technique . Finally, *custom rule-based matching* has been designed for the recognition and extraction of entities and keywords of interest.

A user interface has been designed and developed to put the tool into production. It displays the output of the models graphically and interacts with internal databases to allow the agents in charge of supervising the model performance to introduce corrections to the predictions for later fine-tuning and improvement of the model.

Key words: Natural language processing, machine learning, unsupervised text classification, supervised text classification, custom entities extraction, automating, business improvement, service engineering reports

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

A fundamental part of maritime technology companies dedicated to delivering innovative products, services, and solutions is the work performed by the service engineering department. This department is responsible for installing, maintaining, and inspecting the products that are produced by the company.

It is a tedious but very important task to design a workflow that allows extracting as much knowledge as possible from the reports made by the engineers as these reports contain really valuable information for other departments of the company such as Aftersales, Warranty, Customer Support, Marketing, etc.

A very common problem in companies with a long history in the market is that CRM's (customer relationship systems) do not allow this information to be collected in a structured format, instead service engineers have to upload word documents explaining the work performed and future recommendations in a raw text format with no structure at all. This makes it difficult to analyse these texts for subsequent decision-making.

In big enterprises, making changes to the CRM and these forms is a long-term process and there is a need for a short-term solution to automate this kind of process meanwhile. This is when Artificial Intelligence is needed. The design and deployment of the appropriate backend system and AI technology can potentially save the company a significant amount of time and money.

1.1 Motivation

The company in which I have had the opportunity to work has a customer portfolio of more than 30,000 vessels, more than 7,000 employees and 200 years of experience, so it is essential to have a well-developed internal workflow system that allows optimal collaboration between all departments to optimise processes and tasks of their agents and thus increase profits.

Due to its long presence in the market, the coordination between the service engineers department and the AfterSales department has become a bit outdated with respect to what new technologies allow.

The current process for service report classification is cumbersome and time-consuming. The service reports are sent to a team in India to be entered into

IPDB database. A report is generated at regular intervals (excel file) and sent to members of the life-cycle management team in Croatia. They will then read all the reports, and classify them into the classes of interest such as safety issues, non-conformity, warranty cases, quote requests, among others.. Any reports that require further investigation or registration are then sent to the relevant parties.

This process requires a lot of manpower, and the resources involved are needed for other tasks. Therefore there is a need to propose an innovative solution that could allow the automation of the process of reviewing service reports for their subsequent classification. This solution will use Natural Language Processing techniques to attempt automatic classing of the reports. The resulting analysis will be stored in a database, and a front end will be made for the analytic team to facilitate their work.

During the analysis of the reports, any mention of spare parts used could be forwarded to the relevant spare part team so verification can be made on whether the parts were brought on board by the engineer, or taken from the vessel store. An offer cold then be made to replenish on-board stores if necessary. On occasion a vessel crew will ask the service engineer for a quote on pars needed. This is not always picked up and forwarded by the back office organization, leading to the customer having to contact the company again, to get the quote. This does not give a good impression of the organization. This information could be automatically extracted and forwarded to the relevant spare part team using a custom entity recognition. There are probably more opportunities to be found in the service reports, that are not thought of here, so more investigation could prove useful.

1.2 Objectives

The aim of this project is to automate the internal processes of Kongsberg Maritime S.L.U by applying artificial intelligence in form of a semi-supervised machine learning model which preprocesses raw input texts and classifies them into the classes of interest of the company taking into account all the available information in them. This will project will also raise awareness of the importance of digitization and automation within the business.

The end user of the tool will be the engineering team in charge of reviewing the reports, so the desired result must have a graphic interface that allows this team to read the report, see the prediction, and take any required action on it, for example, sending an email to the marketing department with the report reference. Also, it is important to create a connection to a database so that this team can introduce corrections to the predictions made by the model in order to be able to improve the model by fine-tuning it over time.

With this being said, this objective can be broken down into the following sub-objectives:

- Select the data sources that will serve as input to the models
- Design a factory method pattern in order to keep the data loading process abstracted and easily maintained.

- Study with text preprocessing techniques in order to find the one that suits the purpose the best.
- Decide the number of desired classes depending on how specific they shoud be.
- Determine the data field that better summarises the engineer's work for classification purposes: it could be either **"Work carried out"** , or **"Diagnostics"**, or a combination of them both.
- Explore different text vectorization techniques and unsupervised machine learning models to come up with a classifier that is able to detect the different classes.
- Create a custom rule-based matching for the recognition and extraction of entities and keywords of interest such as Kongsberg products, obsolete products, and safety issue keywords among others.
- Design a graphical interface to display the model output so that the engineering team in charge of reviewing the reports can easily incorporate this tool into their work routine
- Develop a final APP that displays the output of the classifier applied to the incoming reports. This way the model performance will be supervised by the mentioned team for the first months in order to get corrections from them and start generating a set of labeled data to be able to train a new supervised classifier on this data afterwards. Select and fine-tune the supervised model that suits the purpose the best.

1.3 Expected impact

The engineering team that is responsible for manually examining the reports will find this tool to be of tremendous assistance. Once the model performance is stable enough, the following improvements will be noted:

- A reduction of invested time in repetitive tasks and an increase in team productivity by dedicating this saved time to other high-value activities.
- A possibility of a massive search of information in the different documents and further analysis.
- A possibility of analyzing the life cycle of those spare parts that cause the most part of safety or non-conformity issues.
- Increased traceability of the information.

1.4 Memory structure

This report consists of 7 chapters:

- **Chapter 1**, Introduction: This section sets out the motivation, objectives, expected impact, structure of the report, and the collaborations that have taken place in the development of the project.
- **Chapter 2**, State of the art: in this chapter the current state of the art is discussed and the related work is overviewed. The most important background concepts are also explained.
- **Chapter 3**, Data preparation and understanding: this section brings up the dataset description, the preprocessing carried out and the technologies used for dealing with the data and for achieving the desired results throughout the project.
- **Chapter 4**, Analysis of the problem: the proposed solution is detailed and the working methodology followed for the development of the code is explained.
- **Chapter 5**, Experimentation: the experimentation of this project is divided in two sections, the carried out experimentation with all models designs of both parts is detailed in this chapter.
- **Chapter 6**, Extracted Knowledge and Model Evaluation: the review of selected unsupervised techniques from the first part of Experimentation is made. The evaluation of supervised models which design was explained in the second part is explained.
- **Chapter 7**, Conclusions: the objectives that have been achieved are presented, detailing how they have been implemented. This chapter also discusses the limitations faced in different phases of this project and possible areas of improvement are mentioned for the future. Finally, the relationship between undertaken studies and present research is discussed.

Lastly, the bibliography is included with the all the cited bibliographic references and all the sources consulted.

1.5 Collaborations

This final degree work has arisen as a result of a project proposed in the company Kongsberg, so it is necessary to mention all the collaborations that have taken place during its realization. Jan Erik Hjelseth has been my tutor and the person who has been in charge of implementing this functionality in the internal processes, so he is the one who has set the objectives and has supervised the development and deployment of the model. He also performed an exploratory data analysis beforehand to see if the available data was sufficient for the main objective. Andrea Radovic, a project engineer, has been in charge of manually reviewing the output, i.e., the predictions of the first iterations of the model to start generating a correctly labeled dataset for the further development of the supervised model.

CHAPTER 2
State of art

2.1 Background concepts

2.1.1. Artificial Intelligence

Artificial Intelligence (AI) is a branch of computing dedicated to developing intelligent machines that can perform tasks and actions that normally require human intelligence.

The most generally used informal definition of AI was created by Alan Turing. The so-called Turing test [1], [2], [3] is quite easy. Let's say that something that we hide behind a curtain communicates with us. If we are unable to distinguish it from a human, then it is artificial intelligence. It is also common to attribute the birth of this field of research to computer scientist John McCarthy, Marvin Minsky and Claude Shannon. This concept designated at the Dartmouth conference in 1956 [8] was defined as the science and engineering of "making intelligent machines." See Figure 2.1.

Today's era of quick technological development and exponential growth in extraordinarily huge data sets (referred to as "big data") has allowed AI to go from theoretical concept to practical implementation on a previously unheard-of scale.

2.1.2. Machine Learning

Machine Learning is a branch of Artificial Intelligence. It is used in different sectors with different objectives. Machine learning is what lies behind chatbots and text processing apps such as translators, predictive text suggestion apps, etc. It powers autonomous vehicles and image and face recognition applications. The main concept behind the idea is that a computer program can learn and adapt to new data with no human intervention. The latter is possible by algorithms created exclusively for it: machine learning models.

A model is a mathematical construct that finds patterns or makes predictions based on some examples it has learned. This definition consists of three main parts: mathematical structure, prior learning, and prediction.

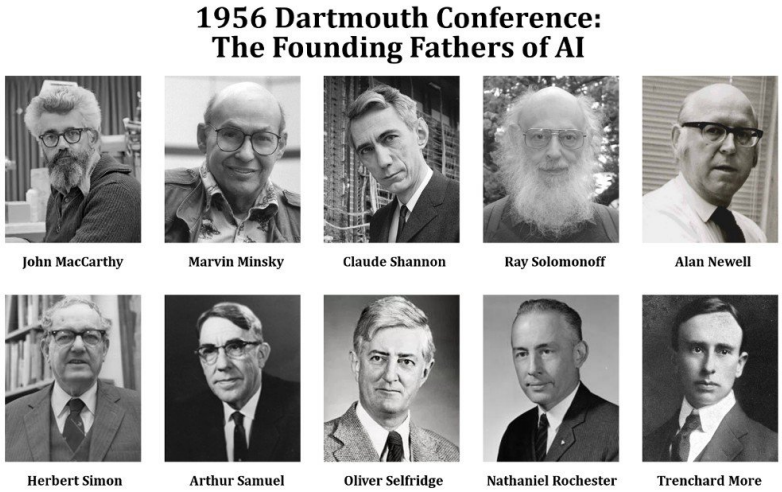

Figure 2.1: Founding fathers of AI. Source: [5]

The models are made of equations, functions, and formulas that combined have the potential to apprehend what occurs inside given information. The variety of their mathematical complexity could be very wide. There are models that exclusively consist of a few addition and multiplication operations, which include linear regression [4]; and models containing millions of trainable parameters, as explained later. Every model must be trained in order to be able to recognize patterns and make predictions on new data.

In what comes to machine learning fields of study, **unsupervised learning** and **supervised learning** are the two primary approaches used in machine learning, although **reinforcement learning** became a very promising branch as well these past years, as shown in Figure 2.2.

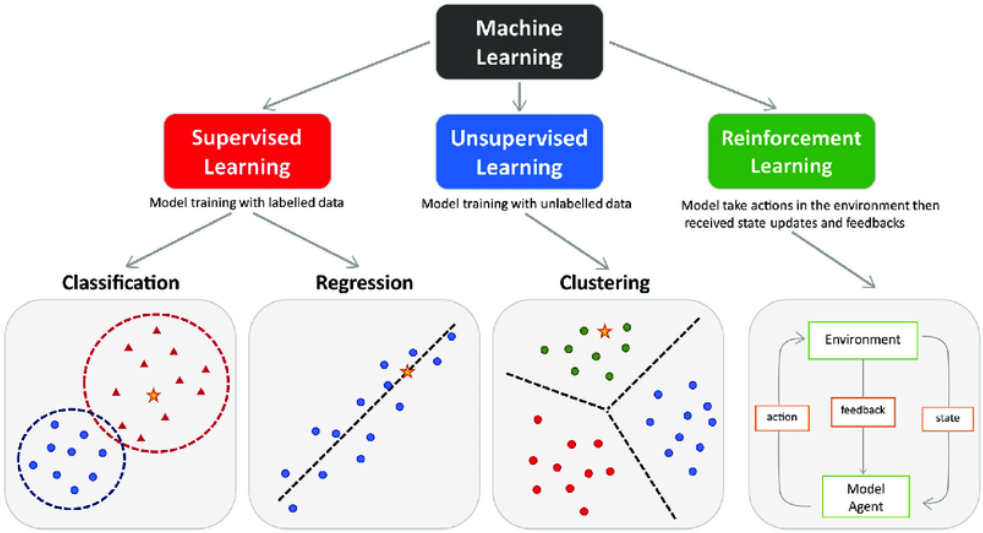

Figure 2.2: The main types of machine learning. Source:[6]

Supervised learning algorithms base their learning on a previously labeled training data set. By labeling it is meant that for each occurrence of the training data set the value of its target attribute is known. This will allow the algorithm to be able to "learn" a function capable of predicting the target attribute for

a new dataset. The two main families of supervised algorithms are regression algorithms when the outcome to be predicted is a numerical attribute and classification algorithms when the outcome to be predicted is a categorical attribute, which is the case of this project.

Unsupervised algorithms are those that train on a collection of data without previously established labels or classes. That is, no target or class value, whether categorical or numerical, is known a priori. Unsupervised learning is used to do grouping tasks, also known as clustering or segmentation, where the goal is to identify clusters of data that are similar to one another.

Few-Shot Learning and Zero-Shot Learning are sub-areas of unsupervised algorithms, meta-learning, primarly used in Computer Vision[computer vision]. It involves categorizing new data when you only have a small number of training samples with supervised data, using the *N-way-K-shot* classification approach [22]. It refers to developing a classification model that is tested on labels it has never seen before (zero-shot) or has only seen a few samples after being trained on a specific set of classes (few-shot). Few-Shot Learning is a very new field that requires additional study and development.

2.1.3. Deep Learning

Deep learning can be defined as a set of Machine Learning algorithms that attempt to model high-level abstractions in data using architectures computational systems that admit nonlinear transformations [13] and try to emulate the behavior of the human brain when learning from data.

Recent developments in Deep learning, that involve the learning of large neural network-style models with numerous layers of representation, can be credited for most of the gains artificial intelligence has made. In a range of tasks involving vast volumes of labeled data, such as image classification [14], machine translation [15], and voice modeling [16], deep learning models have demonstrated excellent performance.

The main difference between Machine learning and Deep Learning lies in their training processes, especially in the feature extraction. In Machine Learning a human agent guides the model on what type of feature to look for, while in Deep Learning the feature extraction is fully automated and no human intervention is needed: see Figure 2.3.

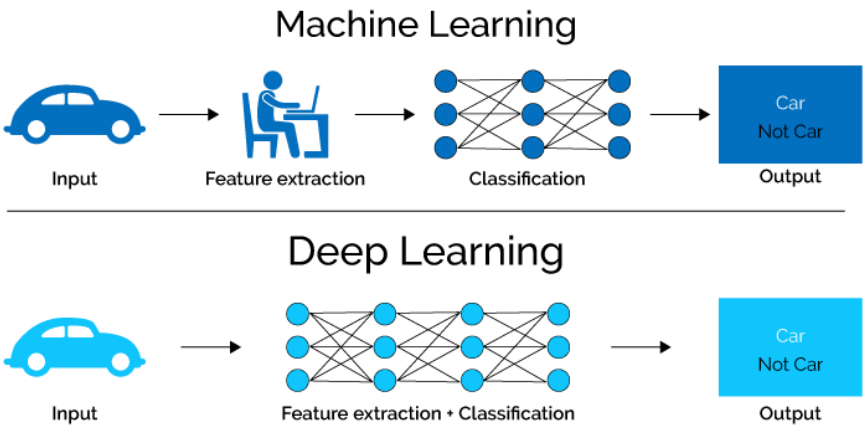

Figure 2.3: Machine Learning vs Artificial Intelligence. Source:[8.2]

A brief description of how Deep Learning models are usually trained will be detailed below. To do so, it is necessary to define some background concepts such as the following:

- **Batch**: the number of samples to work through before updating the internal model parameters, that is, the weights.
- **Epoch**: the number of complete passes through the training dataset. An epoch means that each sample in the training set has had an opportunity to update the model weights.
- **Loss function**: is a function that compares the target and predicted output values; measures how well the neural network models the training data. The cost of this function must me minimized.
- **Optimizer**: is a function, an algorithm or a model that modifies the weights of a neural network.
- **Learning rate**: is a parameter that measures the speed of convergence of optimizing towards a minimum of the loss function.

So a a batch of data is obtained from the training set and entered into the model, obtaining its predictions. The value returned by the loss function is obtained by comparing the predictions with the ground truth. The optimizer then updates the weights of the model in order to minimize the value of the loss function. This processes is repeated for all of the batches of the training set, and once this is done an epoch will have ended. Then all the previous steps are repeated for all the epochs. End of training process.

2.1.4. Natural Language Processing

Natural language processing (NLP) strives to build machines that understand spoken and written human language, natural language. It is a field of study of Artificial Intelligence and a subfield of Machine Learning (See sections 2.1.1, 2.1.2).

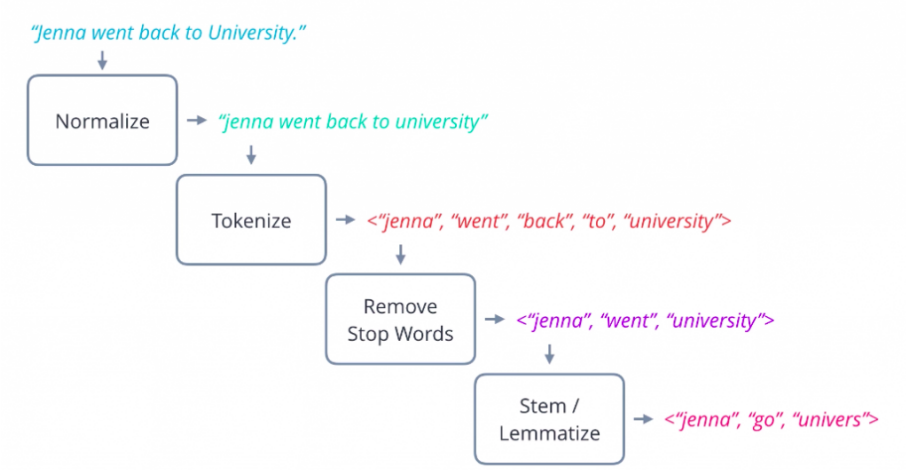

Figure 2.4: NLP pipeline. Source: [9.1]

In addition, the emergence of Deep Learning models like transformer-type neural networks (further explained in the section 2.5) has marked the beginning of a new era in NLP.

One crucial step in every every Natural Language Processing (NLP) project is text cleaning and pre-processing. Since raw data is often filthy, feeding it into a machine learning model could result in a variety of issues, such as imbalance, outliers, noise, redundancy, noise, and incompleteness. To solve these issues, a suitable data pre-processing pipeline is required, one example of it is shown in Figure 2.4. Some of the steps are explained below:

- Tokenization: process of breaking the raw text into small chunks, i.e, words called tokens which are fed to the model.
- Filtering Stopwords: stopwords are common words, such as "a", "and", among others, that are present in the text but generally do not contribute to the meaning of a sentence.
- Lemmatization and Stemming: text normalization techniques that transform a word to its base root mode.
- Part of Speech Tagging: method of classifying words in a text (corpus) in accordance with a specific part of speech, depending on the word's definition and its context.

Once the text is preprocessed and its necessary features highlighted, there are two large families of NLP algorithms to choose from. The first is that of the Machine Learning and Deep Learning models already explained above and the one used in this work. The alternative is rule-based models and systems, which use linguistic rules designed by domain experts. This approach has been used in this work for the entity extraction task.

2.2 Related Work

The related work in the field of unsupervised learning includes the following approaches:

The approach proposed by Dominik Stammbach and Elliott Ash [18] aims to learn to assign classes to the whole dataset without providing ground-truth labels by borrowing an algorithm used in a recent paper for the image classification *Semantic Clustering by Adopting Nearest neighbors* [20]. SCAN has been demonstrated to be effective for image classification but what Dominik and Elliott propose here is that for each document, they get semantically informative vectors from a large pre-trained language model. The hypothesis behind this technique is that neighbors in representation space often share the same label. In some settings, we achieve performance close to a supervised regime. So the conclusion is that documents and their close neighbors in embedding space often share the same class in terms of topical content. Therefore, as with images, unsupervised learning with SCAN can be used for text classification.

Another unsupervised learning method used previously in natural language processing for text classification purposes is the one proposed by Ko and Seo [21] which makes use of keyword lists to categorize sentences directly into a certain number of predefined categories. This approach requires the compilation of such keyword lists, for which expert domain knowledge is needed. The suggested method breaks down the texts into sentences, then uses a sentence similarity metric and keyword lists for each category to categorize each phrase. Following that, it uses the categorized sentences for training, architecture explained in Figure 2.5. This way a training set is automatically generated. The main issue of this method is the definition of these keyword lists, which need further investigation. The experimentation shows that according to the obtained results with respect to performance, the difference between the proposed method and the method by supervised learning is insignificant, only 3.8%.

2.3 Classy Classification

Classy Classification is a Spacy [1] text categorizer for few-shot learning developed by the author DavidFromPandora in the following GitHub repository [25]. This repository belongs to Pandora Intelligence company [24], which is an independent intelligence company, specialized in security risks. Inspired by Hugging Face [2] models for few and zero shot classification and Rasa NLU approach [26] David decided to develop its own approach suitable for using with sentence-transformers [3] or Spacy models. After checking the source code, it is clear that he sets and fits the *Support Vector Machine* model for getting the predictions in each episode, but this model will be explained in the following section.

[1] https://spacy.io/
[2] https://huggingface.co/
[3] https://www.sbert.net/

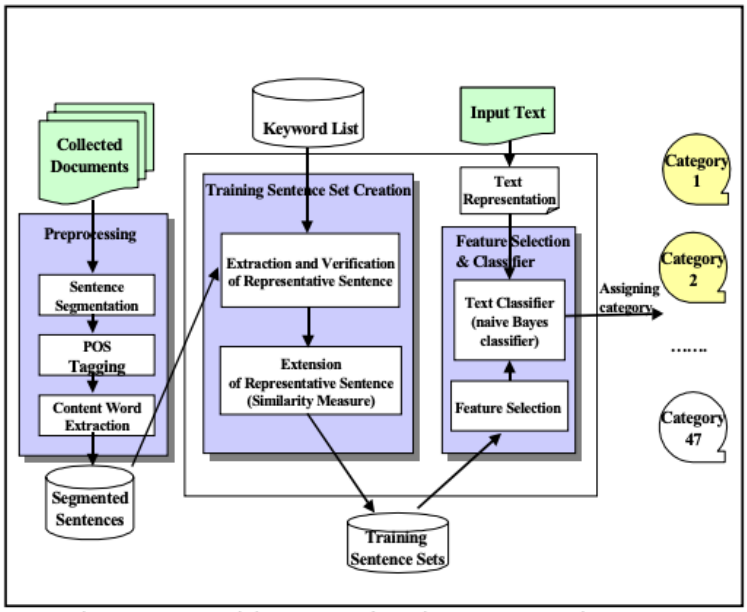

Figure 2.5: Architecture for the proposed system. Source: [11]

2.4 Support Vector Machine

Support Vector Machine has become a widely used tool for classification purposes. It dominated the field of **Supervised Machine Learning** for the past decade due to its outstanding results in comparison to other models. Its decision function is an optimal "hyperplane" that separates observations from different classes based on patterns of information called features [27], and only the calculated support vectors of each of the classes are taken into consideration to determine this boundary. This hyperplane should maximize the margin, i.e., the distance between the support vectors of the two to-be-predicted classes (in binary classification), see Figure 2.6 . This hyperplane is curved in those problems where the features are not linearly separable, that is, in higher dimensionality cases. The margin could be hard, with no training errors permitted and the slack variable is set to zero or a larger one with greater generalizability which allows the classifier to misclassify.

In more difficult classification cases, where outright curved hyperplanes are required and the classifier depends on the data in nonlinear way, some kind of kernel method is typically needed in order to transform the support vectors to a higher-dimensional input space, for instance, polynomial kernel for image processing. These kernel methods could also be used as form of dimensionality reduction for linear SVM tasks.

2.5 Neuronal network

A *Neural Network* is a complex mathematical model that uses interconnected nodes or neurons in a layered structure that resembles the human brain. It creates an adaptive, connectionist system that computers use to learn from their mistakes

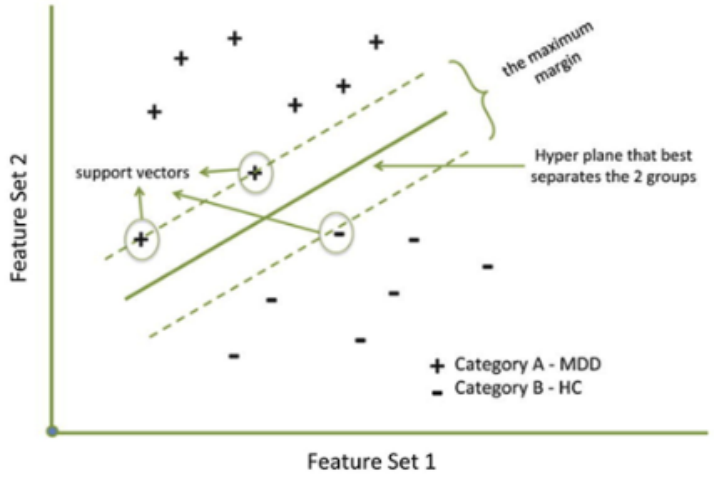

Figure 2.6: Hyperplane that best separates the classes. Source: [27]

and continuously improve. The basic units are neurons, which are usually organized in layers, as shown in the illustration below: Figure 2.7.

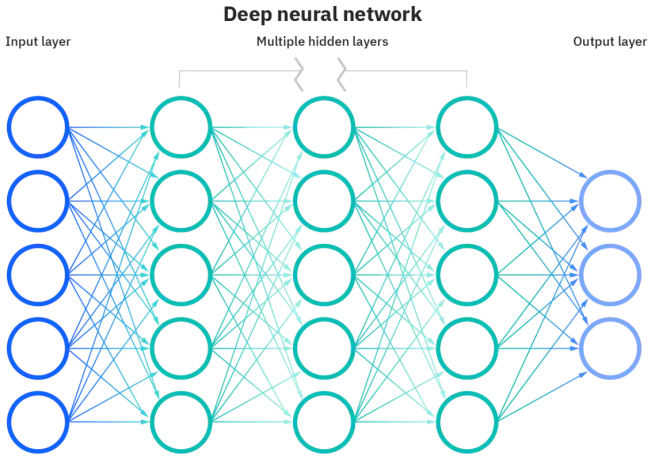

Figure 2.7: Neural Network. Source: [11]

The neurons in this system are linked to one another through links, and the output value of neuron n is multiplied by a weight w and adjusted in accordance with an activation function g. Another neuron in the following layer will use this result as its input value. The trainable values of this model are precisely the weights w mentioned before. An activation function is a function whose goal is to give the network non-linearity. There are numerous types of activation functions, for example, the ReLu function [12] corrects negative values to zero while leaving positive values unchanged. As for the layers, there are three types of layers: input, hidden, and output layers. A network is considered to be fully connected if every neuron in each of its layers is connected to every neuron in each of the layers below it. From the time the data is input until a prediction is issued, the process is called forward propagation, while the process of re-training and updating the w-weights is known as back propagation.

There are numerous types of neural networks in each of the sectors of Deep Learning. Some of these networks are multilayer neural networks, convolutional networks, recurrent neural networks, transformers and graph-based networks.

Transformers

This type of network has been a revolution in many aspects of current Deep Learning. It was proposed in 2017 by Google Brain researchers in the famous paper *Attention is all you need* [7] as a sequence-to-sequence architecture. The main purpose of *Transformers* is to be able to apply a layer of attention that is capable of analyzing several fronts of information at the same time, not one at a time.

Using multi-attention on the input data makes this model perform really well in the fields of natural language processing (NLP) and computer vision (CV). Regarding its disadvantages, this type of networks require a huge amount of data in order to be trained correctly, in comparison to other neuronal network types.

In this project a *Bidirectional Encoder Representations from Transformers* (BERT), in particular BERT_base model, with 12 encoders with 12 bidirectional self-attention heads, will be used with predictive purposes. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google [34].

2.6 Metrics

One of the most crucial parts of the model creation process is this section. The model may appear to be good when it is not, and vice versa, depending on the metric used to evaluate how well the model fits the training data.

Primarily, depending on the type of model, the criteria utilized to evaluate performance are significantly varied.

Multi-label classifiers are commonly evaluated in terms of precision and recall, see Figure 2.8; or using their combination by employing metrics like F-measure. In short, precision tells precision tells how many of the selected objects were correct, see Equation 2.3. Recall tells you how many of the objects that should have been selected were actually selected, as shown in the Equation 2.5.

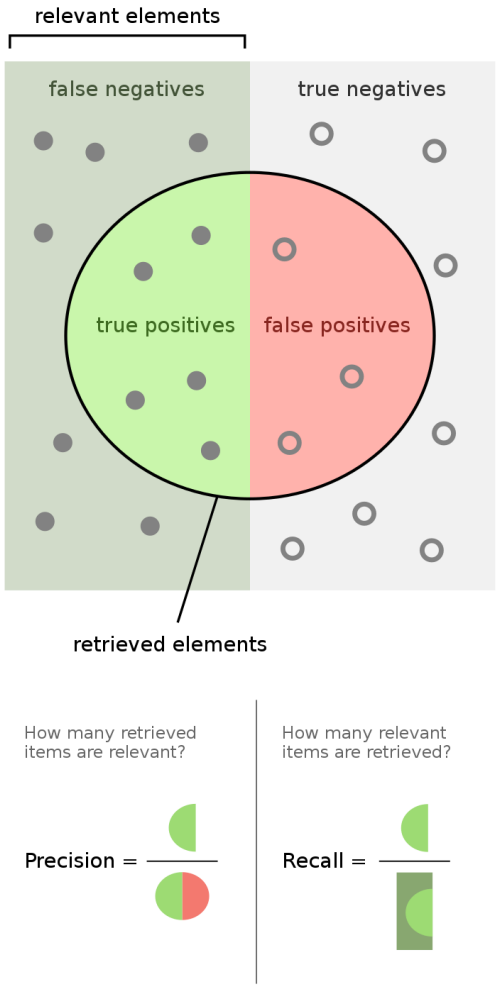

Figure 2.8: Precision vs Recall. Source: [9]

Precision and recall work in opposition to one another, meaning that raising one lowers the other. In the extreme circumstances, precision is very high while recall is very low if you choose virtually everything, and precision is very low while recall is very high if you choose almost nothing. Therefore, achieving some type of balance between the two is the aim. Using the F-1 score, which is the harmonic mean of recall and precision, is the most popular method for doing this.

So F1-macro was the measure used for evaluating all the models during the experimentation with supervised models phase. F-1 macro takes into account the precision and recall of the system's predictions while employing the macro-averaging method to prevent the model to be biased towards the most populous classes.

Another useful metric in classification tasks is the confusion matrix, which is a matrix of rows and columns that evaluates a model's performance. The algorithm predictions are put in the columns, while the actual values are put in the rows, as shown in the Figure 2.9. The primary goal is to increase the amount of samples that fall on the matrix's major diagonal (increase of correct predictions).

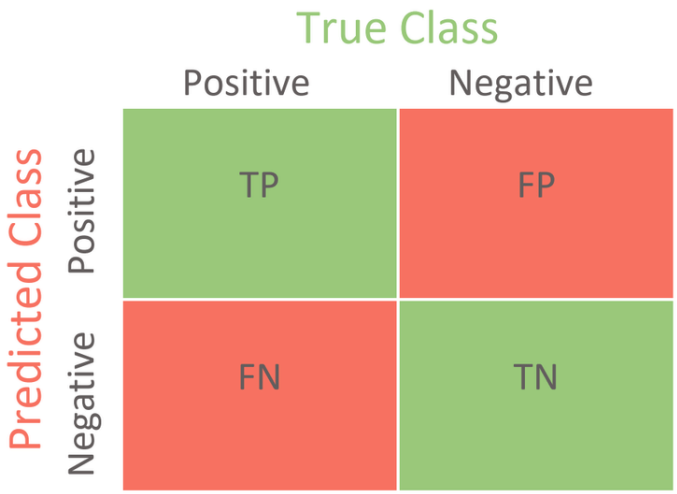

Figure 2.9: Binary Confusion Matrix. Source: [10]

$$F_1 - macro = \frac{1}{|L|} \sum_{l \epsilon L} F_1(y_l, \hat{y}_l) \tag{2.1}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.2}$$

$$precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.3}$$

$$precision = \frac{1}{|L|} \sum_{l \epsilon L} Pr(y_l, \hat{y}_l) \tag{2.4}$$

$$recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.5}$$

$$recall = \frac{1}{|L|} \sum_{l \epsilon L} R(y_l, \hat{y}_l) \tag{2.6}$$

where L is the set of labels, yl the ground truth, and ŷl the predicted labels

CHAPTER 3
Data preparation and understanding

This chapter's main goal is to present the dataset that was utilized in this project, along with their brief summary and technologies used for its handling.

3.1 Technologies used

Before starting to list all the experimentation done, it is necessary to explain which programming language, frameworks and tools have been used to develop this part of the work.

All the code in this work is done in Python3 [1], one of the most widely used programming languages in Data science. Since the experimentation phase has been fairly extensive, many libraries have been used; nonetheless, they can generally be categorized into the following groups, where I will highlight the most significant libraries:

- Data handling libraries: *Pandas* [2]
- Natural Language libraries: *Nltk* [3], *Spacy* [4], *re* [5]
- Mathematical functions and operations: *Numpy* [6], *Statsmodels* [7], *Sklearn* [8], *Scipy* [9] *y Huggingface* [10].
- Data visualisation libraries: *Matplotlib* [11] *and Seaborn* [12]

[1] https://es.wikipedia.org/wiki/Python
[2] https://pandas.pydata.org/
[3] https://www.nltk.org/
[4] https://spacy.io/
[5] https://docs.python.org/3/library/re.html
[6] https://numpy.org/
[7] https://www.statsmodels.org/stable/index.html
[8] https://scikit-learn.org/stable/
[9] https://scipy.org/
[10] https://huggingface.co/
[11] https://matplotlib.org/
[12] https://seaborn.pydata.org/

- Deep Learning: *torch* [13] *and transformers* [14]

Microsoft Access [15]has been used to build the first version of Frontend.

3.2 Dataset

The available dataset has 2326 unique reports written in English with 5 features, i.e. it has the shape of 2326 x 5 although that corresponds only to the data collected from January 2022 to April 2022, so there is more data available in the company database if needed. The features are: **docid** which is a unique id string for each report; **Diagnostics** which is a raw text written by service engineer with the diagnostics of the incidence; **Work carried out** which is also a raw text explaining the work carried out, on average it is a paragraph of at least five long sentences; **Recommendation** which is also a long string of raw text with no predefined values, normally is set to "none noted"; and finally **Parts used** which is a string mentioning relevant information on Spare parts of the incidence. A Dataset sample is shown in the Figure 3.2.

As observed, none of these features have predefined values that are repeated, on the contrary, they are all natural texts. For instance, this is an extract of **Work Carried Out** values, the dates have no common format and there are missing spaces, spelling mistakes and also missing full stops; all this could be seen in the Figure 3.1.

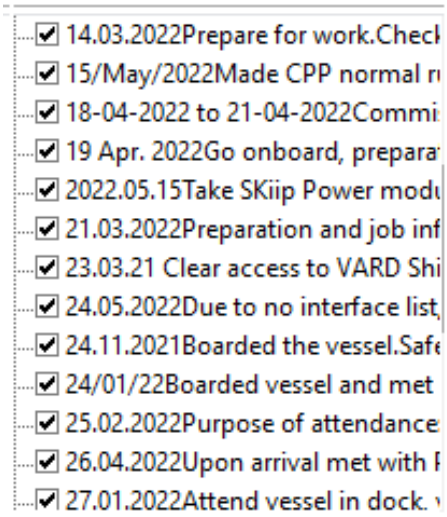

Figure 3.1: Work Carried Out feature values. Source: Own elaboration

[13] https://pytorch.org/
[14] https://huggingface.co/docs/transformers/index
[15] https://www.microsoft.com/es-es/microsoft-365/access

docid	Diagnostics	Work carried out	Recommendation	Parts Used
DB76471CEE486FEEC125	Simens:Still open issue aft	14.02.2022FSE onboard Malme	None noted	None noted
BB8FA9CBD3FC8219C12	Travel to Bilbao to do PCR	23.03.21 Clear access to VAR	Advise shipyard for brake l	None noted
31616A3455EE2895C125	Test and adjust as require	24/01/22Boarded vessel and n	None noted	None noted
9CDA3222F6F48EE6C125	During Operation of Lowe	05.03.2021During Inspection, i	None Noted	None Noted
0E55ED8ECD351E7BC125	The crew did a manouver	25.02.2022Purpose of attenda	None Noted	non return valves, balance valves
096FD20E9BEFB896C125	Tested timer on pms syste	09.04.2022Attendance after s	None noted	None noted
CB0E0882271057F2C125	The chief engineer reporte	21.03.2022Preparation and job	None noted	None noted
A2811985E21451F4C125	As for customer complain	6-04Get aboard. Spare parts c	Oil filters renewal at first or	None Noted
FED395B4DC4684B8C125	Main bridgelever cpp wen	Service Order No: AX18229 M	A quotation of following sp	None Noted
506C6D6F24527F99C125	Test all alarms, found the	27.01.2022Attend vessel in do	None Noted	Cables, steel glands, gasket, thruster, filter

Figure 3.2: Dataset sample. Source: Own elaboration

3.3 Data Preprocessing

The preprocessing was carried out with my own Python script. The aim was to leave the reports' texts in the best condition possible in order to make it easier later to extract meaningful information for the classification models.

As already mentioned in the Dataset Description section 3.2, the main problem with all the variables is that they are natural language texts, so several text cleaning techniques have been applied to remove all unwanted tokens, regular expressions were used in order to eliminate all possible date formats (initially there were more than 50 different ways of writing dates), lowercasing was applied to all the corpus, stopwords were also removed for some of the models [16] and some basic tokenization was also done (such as separating punctuation from words). Some of the used methods are shown in the Listing 3.1.

One of explored preprocessing options was the Python wrapper for Language Tool [17] which is an online style and spell checker for natural language. Although most of the corrections made by it were precise and correct, some acronyms, abbreviations and part serial numbers were corrected incorrectly, which was an undesirable behavior. Therefore, the use of this tool has been discarded.

```
1  def regex(df:pd.DataFrame, column) -> pd.DataFrame:
2
3      #tool = language_tool_python.LanguageTool('en-US', config={ '
           cacheSize': 1000, 'pipelineCaching': True })
4      #df[column] = df[column].apply(lambda x: tool.correct(x))
5
6      rx = r'\.(?=\D)' # to insert a white space after full stops
7      df[column] = df[column].apply(lambda x: re.sub(rx,". ", str(x)))
8
9      rx0 = r'\d{1,2}\w{2}[\s]\w{1,3}[\s][.][\s]{1,2}?\d{0,5}[)]' #to
           grab 09th dec . 20211)
10     df[column] = df[column].apply(lambda x: re.sub(rx0," ", str(x)))
11
12     rx1 = r'\d{1,}[-\/.\s]\d{1,2}[-\/.\s]?\d{0,5}'
13     df[column] = df[column].apply(lambda x: re.sub(rx1," ", str(x)))
14
15     rx2 = r'\d{1,}\w?[-\/.]\w{1,}[-\/.]?\d{0,5}' # to grab those dates
           that have month as a string
16     df[column] = df[column].apply(lambda x: re.sub(rx2," ", str(x)))
```

[16] https://en.wikipedia.org/wiki/Stop_word
[17] https://pypi.org/project/language-tool-python/

```
17
18      return df
19
20
21  def remove_stopwords_stemmer(df:pd.DataFrame, column) -> pd.DataFrame:
22      stemmer = PorterStemmer()
23      words = stopwords.words('english')
24      words.extend(['from'])
25      df[column] = df[column].apply(lambda x: " ".join([stemmer.stem(i)
            for i in re.sub("[^a-zA-Z]", " ", str(x)).split() if i not in
            words]))
26
27      return df
```

Listing 3.1: Preprocessing methods

As for the feature selection, after examining the data in depth, it was decided to use the variable **"Work carried out"** as an input for the algorithms. This variable contains texts of different lengths explaining the work carried out in the given report. All missing values of this variable have been replaced by the string "none noted" to be further classified in the **"Get More Info"** class. Tests have been made to concatenate this feature with some of the others to obtain an input text to the models with more information but that has only added more noise eventually so this has been discarded.

Last but not least, all predictive models (*Classy Classification, Support Vector Machine, Bert*) have been run at the sentence level, due to the fact that the full text of a report almost always mentions all 8 classes, thus it is impossible to classify a report in its entirety in only one of them. That is to say, a part of the preprocessing consisted of creating an auxiliary dataframe with each report separated by its sentences and each one stored with its unique identifier, which turned out to have 64222 rows. This has been done with the help of the *Spacy Sentencizer* pipeline component [18] which has proved to detect sentence boundaries better than the alternative *NLTK tokenizers* [19].

[18] https://spacy.io/api/sentencizer
[19] https://www.nltk.org/api/nltk.tokenize.html

CHAPTER 4
Analysis of the problem

4.1 Proposed solution

This project is divided into two work blocks. The first part has been the one that has required the most experimentation and time, since it has consisted in building an unsupervised classifier and applying it to the set of reports written in natural language and never previously classified. Before moving on to the classification task itself, a work pipeline has been designed that automates the flow of data from source to destination, that is, it takes into account the entire set of processes that convert raw data into actionable answers to business questions.

So the data could be obtained either from SQL Database or CSV file, this data collection process is abstracted by designing a DataLoader method that is based on Factory Method Design pattern, which follows the "single resposability" principle so that the program is easy to maintain and complex conditional code is avoided, this is explained in more detail in Appendix A. New data sources will be added as the project goes on and the requirements change, so this implementation facilitates the future maintenance.

Once a unified dataframe is created with the data coming from whichever source, a preprocessing pipeline is applied to it. All the reports are split into sentences, the raw text is cleaned and the final dataframe is fed into model. Being a multiclass classifier, the predictions show a probability distribution of belonging to one of the classes and only the two most probable classes are saved.

On the other hand, a custom entity recognition is applied on report level and the result is saved as a html file with all the entities identified and marked with color blocks directly on the text, this functionality will be shown in the final APP in order to facilitate long text comprehension.

Once better results have been achieved than those of a naive [1] model, the model has been put into production generating predictions which have been supervised by Kongsberg engineers, which has allowed the generation of a labeled dataset which has been used to choose and train a supervised learning model.

[1] https://www.oreilly.com/library/view/budgeting-basics-and/9780470389683/9780470389683_naive_models.html

4.2 Methodology

Azure DevOps, specifically Azure GIT Repos [2], was used during development for version control and testing. The Integrated Development Environment (IDE) used in this project is Visual Studio Code [3] with the required extensions to integrate it with Azure DevOps.

The intended branch strategy is shown in the Figure 4.1. In brief, the Master Branch is kept and updated throughout life cycle. This branch will have the latest updates to go into production. And will be used to branch new production versions. The Development Branch is also kept and updated throughout the whole project life cycle and this branch is base branch for all feature and bug fix branches. This branch will have all changes merged to it and tested before merging to Master Branch. Then, every time a new feature is developed a new Feature Branch will be created, branched from Development Branch. They are merged back to Development Branch when the feature is completed and tested. This branch will be deleted after development have been tested and merged to master.

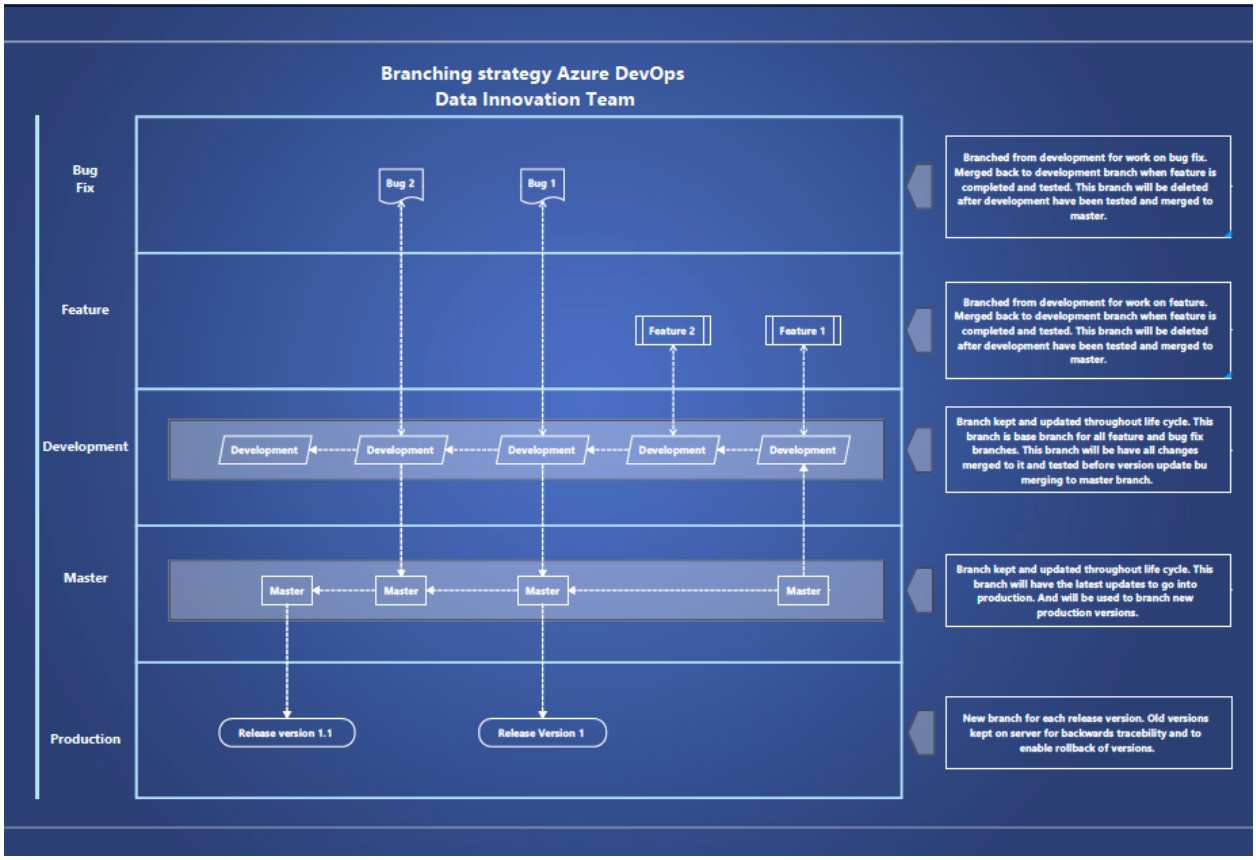

Figure 4.1: Branch Strategy. Source: Jan Erik Hjelseth elaboration

[2] https://azure.microsoft.com/es-es/services/devops/repos/
[3] https://code.visualstudio.com/

CHAPTER 5
Experimentation

Finding the model and hyperparameter combination that best solves the aforementioned challenge is the major goal of the experimentation in this project. It consists of two parts, each in a different scenario. The first part will concentrate on developing an unsupervised multi-class classifier by exploring several methods, including Transfer learning, Few shot learning, Topic Modeling, among others. The task was faced exploring different techniques with the aim of getting better results. Also the custom entity ruler matcher was developed during this phase.

The second phase of experimentation will focus on building a supervised multi-class classifier for the same purpose but using the corrected predictions of the first iteration of the unsupervised model as training set, which results will be evaluated in the following chapter.

5.1 First part: Topic Modeling, Few Shot Learning, Entity Ruler

The first explored reports classification approach was **Topic Modeling**. Bayesian topic modeling [32] is a technique that is related to unsupervised classification. Topic models differ from classification in that the documents are not intended to be assigned a single class label. As opposed to this, topics in topic models are a probability distribution over words, while documents are a probability distribution over subjects. Latent Dirichlet Allocation (LDA) is one of the algorithms used to discover the topics that are present in a corpus. The algorithm takes as input a bag of words matrix (i.e., each document represented as a row, with each columns containing the count of words in the corpus). The aim is to approximate document-topic and topic-word distribution. The number of topics parameter must be specified.

5.1.1. Topic Modeling LDA with *Scikit Learn*

Latent Dirichlet Allocation (LDA) is a probabilistic graphical model based on online variational Bayes algorithm so it only requires raw counts of each word in

the corpus, so a *CountVectorizer* [1] is used to transform the input data. Some manual testing functions were made in order to come up with the best values of its hyperparameters, these are:

- Min-df and Max-df: When building the vocabulary ignore terms that have a document frequency strictly lower and upper than the given threshold, respectively.
- Analyzer: Whether the feature should be made of word n-gram or character n-grams.
- N-gram range: The lower and upper boundary of the range of n-values for different word n-grams.

Gridsearching was also done for *LatentDirichletAllocation* model parameters as well, for learning decay and number of topics especially, the metrics used in order to evaluate each model configuration was Log Likelihood Score.

The final configuration ended up being as shown in the following Listing 5.1

```
1
2  vectorizer = CountVectorizer(analyzer='word',
3                               min_df=10,      # Eliminating words that
                                    appeared in less than 2 documents
4                               max_df=0.80,    # Ignore words appeared in
                                    80% of the documents
5                               ngram_range=(1,2),
6                               max_features=50000,  # Max number of uniq
                                    words
7                              )
8
9  data_vectorized = vectorizer.fit_transform(docs)
10
11 # Build LDA Model
12 lda_model = LatentDirichletAllocation(n_components=5,
13                                        # Number of topics
14                                        max_iter=10,
15                                        # Max learning iterations
16                                        learning_method='online',
17                                        # use mini-batch of the training
                                              data for each update
18                                        random_state=100,
19                                        # Random state
20                                        batch_size=128,
21                                        # n docs in each learning iter
22                                        evaluate_every = -1,
23                                        # compute perplexity every n
                                              iters
24                                        n_jobs = -1,
25                                        # Use all available CPUs
26                                       )
27 lda_output = lda_model.fit_transform(data_vectorized)
```

Listing 5.1: LDA by Scikit Learn configuration

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Although LDA is a very powerful algorithm for topic modeling tasks, it worked poorly with this data, being unable to differentiate the topics within the corpus, as reflected in the Figure 5.1 that shows the words that most contribute to each identified by the model topic.

```
Topic #0 :
test system trial connect sea check sea trial work control new
====================
Topic #1 :
pitch pressur control valv bar work check port test adjust
====================
Topic #2 :
thruster load drive set adjust calibr check gener bow test
====================
Topic #3 :
seal shaft thruster propel new oil box ring blade hub
====================
Topic #4 :
steer gear oil seal pump steer gear port bear replac new
====================
```

Figure 5.1: Keywords of each topic by LDA *Scikit Learn*. Source: Own elaboration

Apart from LDA by Scikit Learn algorithm, *Gensim's LDA* model was also implemented (there was an extensive experimentation process behind this as well but the details will be omitted as the results are not good and this option was discarded). As it is shown in the Figures 5.2 and Figure 5.3 the representative keywords of each topic are not helpful to differentiate the topics either:

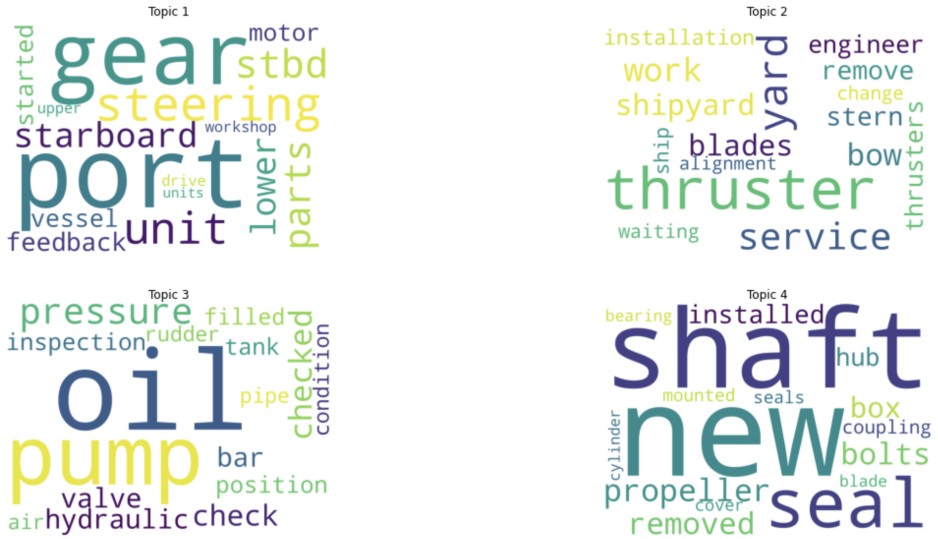

Figure 5.2: Wordcloud by Gensim's LDA of. Source: Own elaboration

	Topic_Num	Topic_Perc_Contrib	Keywords
0	0	0.7123	port, gear, steering, unit, stbd, parts, starboard, lower, feedback, vessel
1	1	0.7714	thruster, yard, service, work, bow, blades, shipyard, stern, remove, thrusters
2	2	0.9394	oil, pump, pressure, checked, check, valve, bar, hydraulic, inspection, position
3	3	0.8492	new, shaft, seal, propeller, removed, bolts, box, installed, hub, coupling
4	4	0.8428	pitch, test, tested, control, adjusted, checked, set, sea, start, time

Figure 5.3: Topics and their keywords Gensim's LDA. Source: Own elaboration

5.1.2. Topic Modeling with BERTopic

A transformer-based models such as BERT show amazing results in different fields of [33], so it was crucial to test it on this data as well. *BERTopic* library [2] was used for the purpose. The mechanism behind it is the following: the model creates a representation vector for each document, then the Uniform Manifold Approximation and Projection (UMAP) algorithm reduces the dimensions that of each vector, the Hierarchical Density-based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm is used for the clustering, after that, the class based Term Frequency–Inverse Document Frequency (TF-IDF) algorithm retrieves the most relevant words for each topic and finally the Maximize Candidate Relevance algorithm is used. This way topic representations from clusters are created.

This model receives a list of preprocessed reports(as explained in Data Preprocessing section 3.3) as input and generates topics and their probabilities, the number of topic is a decision of the model, although after examining the result this parameter can be changed to another number that suits better the data. *BERTopic* can be used with different embedding models. Word embedding is a term used to describe how words are represented for text analysis. It typically takes the form of a real-valued vector that encodes the meaning of the word, with the expectation that words that are adjacent to one another in the vector space will have similar meanings. These models can be selected from Sentence-transformer or from Hugging Face transformers models. In this project, the default *BERTopic* embedding, *"all-mpnet-base-v2"* from Sentence-transformer embedding model and Spacy embedding models were used with different n-grams ranges and neither of them achieved good results. These configurations are shown in the Listing 5.2.

```
# Default Bertopic configuration
topic_model_general = BERTopic(language="english",
    calculate_probabilities=True, verbose=True)
topics, probs = topic_model_general.fit_transform(docs)

# SentenceTransformer embedding model
sentence_model = SentenceTransformer("all-mpnet-base-v2", device="cpu")
topic_model = BERTopic(embedding_model=sentence_model, verbose=True)
topics, probs = topic_model.fit_transform(docs)

# Spacy embedding
```

[2] https://maartengr.github.io/BERTopic/index.html

```
nlp = spacy.load("en_core_web_md", exclude=['tagger', 'parser', 'ner',
    'attribute_ruler', 'lemmatizer'])
topic_model_nlp = BERTopic(embedding_model=nlp, verbose=True)
topics, probs = topic_model_nlp.fit_transform(docs)
```

Listing 5.2: Different Bert configurations

In the following figures [5.4, 5.5] I show "Topic Word Scores" charts by the default BERTopic configuration and BERTopic with spacy embeddings configuration. It is clear that these results are not useful for the purpose at all.

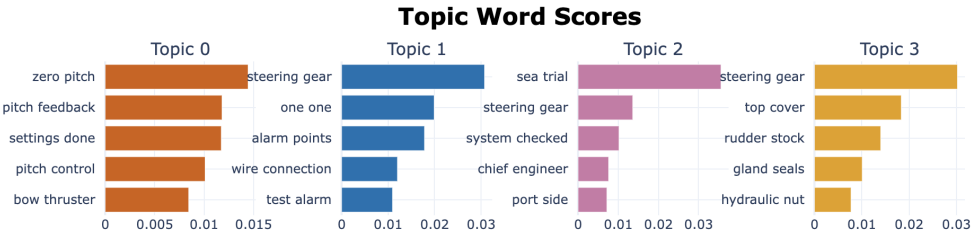

Figure 5.4: Topic Word Scores output by the default BERTopic model with n-grams range of (1,2). Source: Own elaboration

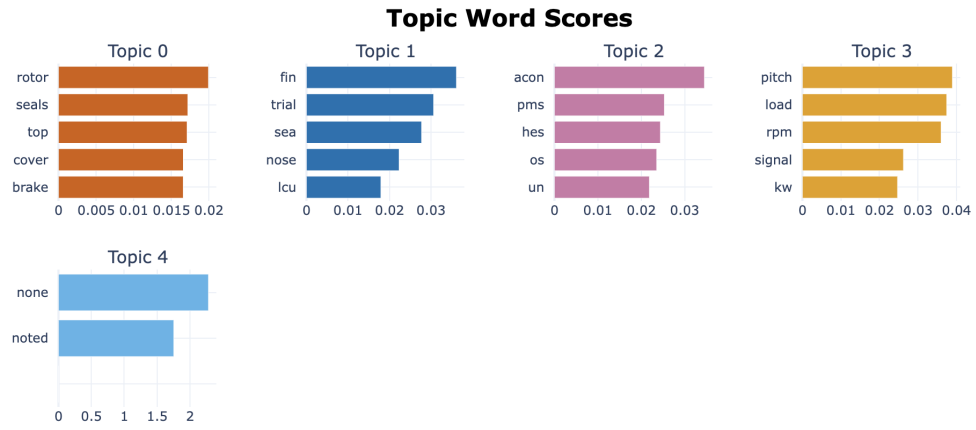

Figure 5.5: Topic Word Scores output by BERTopic model with Spacy embeddings with unigrams. Source: Own elaboration

Furthermore, taking a look at the topic info table of the model with spacy embeddings and n-grams range of (2,3), it can be observed that 797 observations are classified as topic -1, while topic -1 refers to all outliers and should typically be ignored (Figure 5.6). The distribution pattern over the classes was observed with all other configurations. This shows that these models are unable to understand the data.

All of the tested configuration of BERT based models have been shown to work poorly for this specific collection of documents. The reason behind this is that the reports are very technical texts in the engineering field and all deal with the same topic: failures, maintenance, and spare parts in merchant ships

	Topic	Count	Name
0	-1	797	-1_steering gear_shaft seal_bow thruster_od box
1	0	79	0_morning meeting_upload new_tested ok_un un
2	1	77	1_sea trial_cm ok_pitch feedback_max load
3	2	74	2_steering gear_top cover_gland seals_rudder s...
4	3	72	3_none noted_noted none noted_noted none_none ...

Figure 5.6: Topic info BERTopic with Spacy embeddings. Source: Own elaboration

and offshore installations. This is why the next approach to report classification will focus on using few-shot learning for building multi-class classifier.

5.1.3. Few-Shot Learning: Classy Classification

Classy Classification is a few-shot learning type of algorithm which requires a support set of training examples, as already explained in section 2.3. In this phase it will be used for building a multi-class classifier on the available data. All reports will be split into sentences and the model will be executed at the sentence level returning as output the probability distribution over the classes.

It goes without saying that most of the experimentation carried out with this algorithm has been devoted to building a support set suitable for training, which was an iterative process of trial and error and manual prediction checks to evaluate the performance of the model. To avoid ambiguity, it was important to select sentences that are both medium in length and highly precise for each class. Also this model is the only one that turned out to work better with preprocessed texts but without stopwords removal and port stemmer steps.

Choosing the number and titles of the classes in this phase was another crucial choice. The optimal number of classes for the purpose has turned out to be 8, and the following names will be used to identify them: **Safety Issue, Non-Conformity, Warranty, Spareparts, General Reporting, Positive, Quote request, and Get more information**. This conclusion was reached after a data exploration and multiple model iterations. **Safety Issue** and **Non-Conformity** are the most important classes for the business to detect. The support phrases were manually selected by reading the reports and extracting phrases that best match each class. A sample of this support set is shown in the Figure 5.7 and the amount of the suppport sentences per each class in shown in the Table 5.1.

Sentences							
Safety Issue	Non-conformity	Spareparts	Warranty	General reporting	Get More Info	Quote request	Positive
system check after incident	was reading incorrectly	Installed new gland o-rings	warranty work	paperwork for access and work	See attached pdf	customer has ordered	normally.
vessel encounter total black out (dead ship)	non conformity	PAL1 lub oil transmitter	owner claim	mandatory COVID test with ship nurse.	please refer to pdf	is obsolete and need to upgrade	seemed to be function ok
this event happens violently	troubleshooting system	input shaft seal	Updated software	discuss the job plan. Drive back to hotel.	none noted	overhaul of units is required	is normal . travel from
troubleshooting after accident	root cause analysis	Bearings	upgrade system	yard	none noted service none noted	docking	new system behaved normally during voyage
pitch went full astern while maneuvering	chief engineer complained	Bushing	adjustment after sea trial	Drydock not empty of water	self-assessment report service see attached document for details	keep critical spares of HCX3 onboard	Found no leak visually during operations.
fails to respond when given order signal	wrong type of parts installed	OD-Box	warranty claims	preparation of drawings, hand tools and personal belongings	none noted commissioning none noted	recommended critical spare parts to have on board	There were not found any cracks on the insulators inside
impossible to operate the thruster in CCW direction	wrong part delivered	Promas	inspection guarantee	vessel was move out for another location	tests carried out according client sea trial plan	outdated equipment and components that are approaching the end of their lifecycle	was calibrated on the captain's instructions . tested ok.

Figure 5.7: Support Set for Classy Classification. Source: Own elaboration

Safety Issue	38
Non-conformity	25
Spareparts	28
Warranty	25
General reporting	38
Get More Info	12
Quote request	16
Positive	23

Table 5.1: Number of support sentences per class. Source: Own elaboration

Some alternatives to this algorithm have also been tested, for example, *Classy Classification* using pretrained facebook bart model underneath *"facebook/bart-large-mnli"*. The results were not good enough so these other options were eventually discarded and will not be discussed.

5.1.4. Rule-based named Entity Recognition

Part of this project has consisted in developing a named entity recognition to detect and mark Kongsberg Maritime products in the text of reports. After testing several options, the tool that has proven to be the most useful for the purpose is Spacy's *Entity Ruler*[3]. It allows to add spans to the default *Spacy* entities using token-based rules or exact phrase matches. This way a custom rule-based entity recognition system is created at the report level.

First, all the Kongsberg products were extracted from an existing excel file and put into json in form of rules. See Listing 5.3

```
[
{"label": "KM_PRODUCT",
  "pattern": [
  {"lower": "k-fleet"},
  {"lower": "spares"}]},

  {"label": "KM_PRODUCT_OBSOLETE",
    "pattern": [
    {"lower": "fanbeam"}]},

  {"label": "KM_PRODUCT_OBSOLETE",
    "pattern": [
    {"lower": "switchboard"},
    {"lower": "longva"}]}
]
```

Listing 5.3: Json file structure with defined rules

Apart from products entities, a list of 500 keywords related to safety issues and concerns were defined by hand and the same procedure was done in order to create rules for safety and concerns entities (such as "alarm", "deterioration", "damage", "wreck", "fail" etc).

[3] https://spacy.io/api/entityruler

Spacy's *"en_core_web_sm"* [4] trained pipeline for English was used as the basis for the entity rule with most of its predefined components disabled and the overwrite_ents parameter enabled. Then json files with customized rules have been added as pipelines to the created entity ruler to match the desired patterns. See Listing 5.4.

```
# Loads json files with pattern configuration
all_patterns = loader.load_patterns()

# Entity ruler configuration
ruler_config = {"overwrite_ents": "true"}
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser','
    attribute_ruler', 'lemmatizer'])
ruler = nlp.add_pipe("entity_ruler", name= "concern_ruler", config=
    ruler_config).add_patterns(all_patterns['concerns'])
product_ruler = nlp.add_pipe("entity_ruler", name="product_ruler",
    config=ruler_config).add_patterns(all_patterns['products1'])
safety_ruler = nlp.add_pipe("entity_ruler", name="safety_ruler", config
    =ruler_config).add_patterns(all_patterns['safety'])
```

Listing 5.4: Entity ruler configuration

After examining the performance of the created system, it was noted that sometimes mismatching was produced. For example, "drive" is one of KM products but it is also a verb, in which case it should not be matched. This was corrected by looping through the list of matched entities for each report and deleting those cases when the matched entity is of "KM_product type" and the token's part of speech is different from noun. Listing 5.4.

```
doc = nlp(x)
incorrect_tokens = []
for token in doc:
    if token.ent_type_ == 'KM_PRODUCT' and token.tag_.startswith(('R',
        'V', 'A')):
    #KM products should always be Nouns.
        incorrect_tokens.append(token.vector_norm)

ents = list(doc.ents)
for ent in ents:
    if ent.vector_norm in incorrect_tokens:
        ents.remove(ent)
ents = tuple(ents)
doc.ents = ents
```

Listing 5.5: Fixing wrong entitites

The output for each report were saved in a separate table in html format with *Spacy's displaCy visualizer* [5]. The Figure 5.8 shows one of the generated html files.

[4] https://spacy.io/models/en
[5] https://spacy.io/usage/visualizers

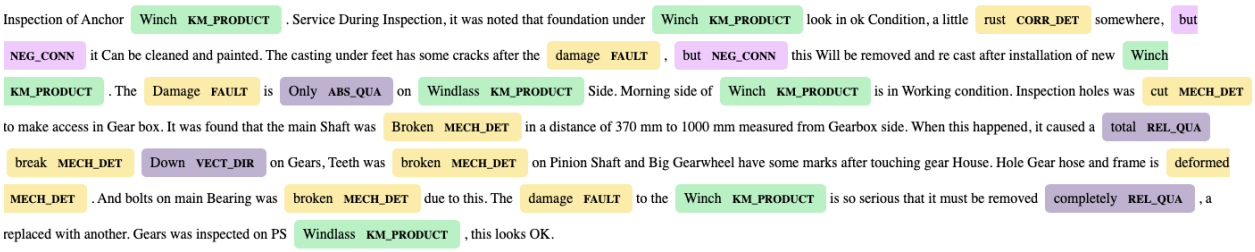

Figure 5.8: HTML output after Entity Ruler. Source: Own elaboration

5.2 Second part: Support Vector Machine and Bert

Getting corrected the predictions from the first iteration of *Classy Classification* Model has lead to a correctly labeled dataset which is going to be used to train a supervised model for the future iterations.

The X feature will still be **"Work carried out"** texts split into sentences and feature Y will be the class corrections for each of the sentences received from the members of the life-cycle management team.

5.2.1. Support Vector Machine

In the case of Support Vector Machine the experimentation consisted in choosing the kernel type to be used in the algorithm and the regularization parameter. The optimal values turned out to be a "linear" kernel and the regularization parameter equal to 1. This is why Scikit Learn LinearSVC [6] module was chosen to train the classifier, which is pretty much the same as Scikit Learn's SVC with parameter kernel set to "linear" but better optimized for a large dataset.

This class expected a numerical input for both X and Y features, thus a transformation of texts was needed. LabelEncoder transformer from Scikit Learn preprocessing module [7] was used in order to encode target values, i.e. the Y feature containing strings with class names.

In order to encode the sentences themselves, Term frequency-inverse document frequency (TF-IDF) Vectorizer was chosen, which converts a collection of raw documents to a matrix of TF-IDF features, i.e. usable vectors for the model. It gives the rare term high weight and gives the common term low weight [28].

The parameters chosen for the Vectorizer are as follows: Listing 5.6

```
# Encode X feature (sentences) as tfidf vectors
tfidf_vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=3,
                        ngram_range=(1, 2),
                        stop_words='english')
X_num = tfidf_vectorizer.fit_transform(X_text).toarray()
```

[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

[7] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

Listing 5.6: TdIdfVectorizer

A further necessary step before training the model is to correct the class imbalance in the labeled dataset, to avoid future biases in the predictions due to the distribution.

Class frequencies prior to oversampling: Counter(0: 1178, 6: 257, 2: 243, 3: 138, 5: 75, 1: 28, 4: 23, 7: 10)

Synthetic Minority Oversampling Technique method from *Imbalanced Learn* library [8] was applied in order to fix the class frequencies. This is a type of data augmentation [29] for the minority class which basically consists in synthesizing new examples for the minority classes. Regarding its parameters, the chosen strategy is "all" which means that all classes will be resampled and the number of nearest neighbours used to construct synthetic samples is set to 3.

Class frequencies after SMOTE oversampling: Counter(0: 1178, 3: 1178, 5: 1178, 2: 1178, 6: 1178, 1: 1178, 4: 1178, 7: 1178)

5.2.2. Bert

Most transformer models have been trained on large amounts of raw text in a self-supervised way, that is, the training data has been labeled autonomously, without help from experts. However, for concrete tasks like this, the general pretrained model has to go through a process called Transfer Learning [30]. Transfer learning involves using a neural network that has been trained for a particular activity to create a second network for a similar task. In other words, it involves utilizing already developed characteristics and learning to create the starting point of a new network created to complete a task associated with the first network. One of the factors contributing to transformers' recent success is transfer learning.

So BERT model is going to be retrained with our labeled dataset, freezing its base and retraining only the last layers. This process and the fine-tuning process will be performed using *PyTorch*, which is explained in broad strokes below.

The initialization of the model is done with the following parameters:

# Initiation of params.	
RANDOM_SEED	58
MAX_LEN	200
BATCH_SIZE	6
NCLASSES	8

Table 5.2: Initialization BERT

"Bert-base-cased" is the chosen model for the purpose downloaded with help of *transformers* library from HuggingFace repository, it distinguishes between lower

[8] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

and upper case which intuitively should work better with this data since it contains many acronyms and proper names.

Tokenization breaks the raw text into words, sentences called tokens. The tokenizer used here is used is the one imported from the pretrained model itself. This tokenizer is defined in a way that is also adds special CLS and SEP tokens to each of the phrases, activates truncation if needed and sets the attention mask as the model requires. Then `MyDataset` class is created in a way that it inherits from the *Dataset* PyToch class and it will be used to create a Pytorch type of dataset out of both training and test partitions.

PyTorch also requires creating a `DataLoader` method that inherits from the *DataLoader* PyToch class and uses `MyDataset` class in order to create batches of data (of size 6 in this case).

The next step is to create the model itself. A `BertClassifier` class is created by inheriting from *nn.Module* Pytorch module, which basically means that it will have all the features that a neural network has. The model is configured then in a way that the first layer is set to a pretrained Bert model, then a dropout layer with p=0.35 is added to drop some neurons during the training process to make the model more capable of generalising. Finally on top of that, a linear neuronal network layer from module *nn.Linear* is added. It is important to assure that the number of output neurons of BERT layer matches the input number of *nn.Linear* layer, which is 768. The number of ouput neurons will be equivalent to the number of classes, that is, 8.

Then it is necessary to create another function, a `forward` method, which dictates how the previously explained layers have to be connected and how the data goes through the system.

Now that our custom top layers are added, the whole model should be retrained. The number of training iterations, i.e. epochs used is 5 and the selected optimizer is *AdamW* with a learning rate of 2e-5. Finally a scheduler is added which aims to reduce a learning rate through the iterations while the model is being trained, in order to optimize the training process. Finally the loss function is set to *CrossEntropyLoss()* which has to be minimized through the iterations.

The developed code is included in the APPENDIX B.

CHAPTER 6
Extracted knowledge and model evaluation

The experiments conducted and described in the preceding chapter are reviewed in this part.

6.1 First Part

Due to the fact that the first part of experimentation was about unsupervised techniques, the tested models are not quantitatively compared with each other. To summarize the experimentation carried out in the first part, all of the tested Topic Modeling techniques have failed to understand the internal structure of the data and have not been able to extract distinguishable topics. However, *Classy Classification* has been the selected model for the first iteration of predictions. Regarding the front-end part, a form was designed in Microsoft Access Software listing all the reports, and once one of them is selected, it also lists its sentences with their predictions and a drop-down list for each of them in case it needs to be corrected. The sentence is coloured in red when the first prediction is **Safety Issue** and in yellow when the first prediction is **Non-conformity**. This tool will be used by life-cycle team's engineers to supervise the performance of the *Classy Classification* model and correct its predictions. In addition, the form is designed in a way that each time a prediction is corrected, the correction is saved in another dataframe which will be used for a supervised classifier later. See Figure 6.1:

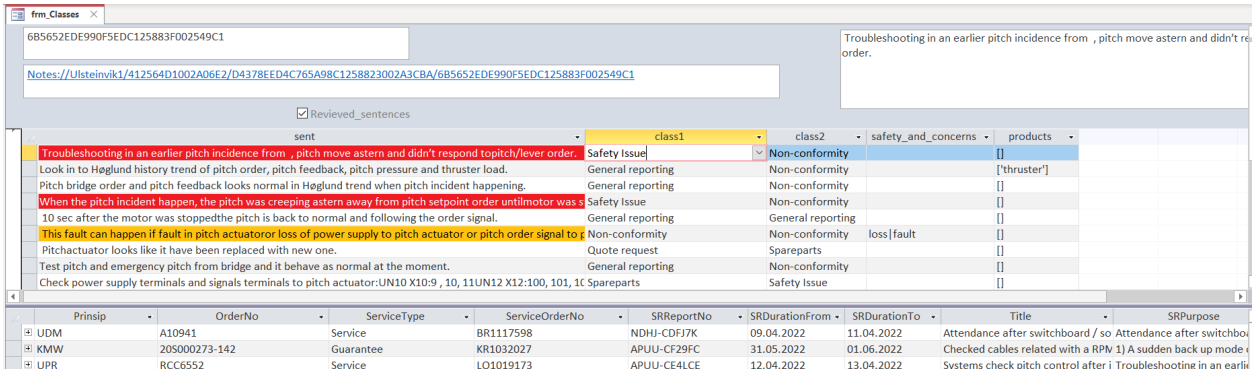

Figure 6.1: Classy Classification predictions shown in Microsoft Access Form. Source: Jan Erik Hjelseth elaboration

The obtained predictions after the first iteration were mostly correct as it turned out after performing their manual correction.

A final feature successfully developed during this first part of the experimentation has been the Custom Rule-based matching system, which has been described in the previous chapter. All the extracted entities by this matching system from each sentence are shown in respective columns as well in the designed Microsoft Access form as shown in the Figure 6.1.

6.2 Second part

The evaluation of results obtained in the Second Part of Experimentation Chapter is explained below, more precisely the evaluation of Support Vector Machine classifier and BERT classifier.

A detail worth mentioning, is that the LinearSVC model has been selected for the purpose due to the experimentation that has been designed initially comparing the performance of several popular models such as RandomForestClassifier, MultinomialNB and LogisticRegression. Cross Validation of $\boxed{CV = 5}$ was used and as illustrated in the Figure 6.2, out of all models, LinearSVC performed the best in all 5 folds, reaching a 0.98 F1-macro metric in one of them.

```
F1 scores of cross_validation [0.83644202 0.81330255 0.8924812  0.66541823 0.47963974 0.39167862
 1.         0.66503268 0.38384906 1.        ]
Mean of cross_validation f1 scores 0.7127844111411539
F1 score Test 0.8350880470016635
```

Figure 6.3: CrossValidation with K-fold. Source: Own elaboration

	model_name	fold_idx	f1_macro
0	RandomForestClassifier	0	0.126846
1	RandomForestClassifier	1	0.120988
2	RandomForestClassifier	2	0.132416
3	RandomForestClassifier	3	0.136505
4	RandomForestClassifier	4	0.093990
5	LinearSVC	0	0.982088
6	LinearSVC	1	0.798333
7	LinearSVC	2	0.919340
8	LinearSVC	3	0.851966
9	LinearSVC	4	0.493817
10	MultinomialNB	0	0.299423
11	MultinomialNB	1	0.324659
12	MultinomialNB	2	0.289329
13	MultinomialNB	3	0.321439
14	MultinomialNB	4	0.162679
15	LogisticRegression	0	0.524948
16	LogisticRegression	1	0.461351
17	LogisticRegression	2	0.448918
18	LogisticRegression	3	0.432243
19	LogisticRegression	4	0.182219

Figure 6.2: Supervised models performance comparison. Source: Own elaboration

The labeled dataset was split into training and testing sets, assigning 80% of data observations to the former and the remaining 20% to the latter.

In order to evaluate the performance of the model K-Fold CrossValidation, which is a statistical method used to estimate the skill of machine learning models by splitting a given data sample into the number of groups (K), switching them to train and test the model and then taking the average score. It was performed with the parameters of K=10 and scoring = "F1_macro", obtaining the following results: Figure 6.3.

The classification report shows very good results as well, especially for the **Positive** and **Quote Request** classes, as indicated by their F1-macro values. See Figure 6.4.

Finally, a normalized confusion matrix is plotted in order to evaluate the model performance graphically. As expected, most of the elements from the main diagonal are close to 1 or equal to 1, which is a sign of a good performance. Figure 6.5.

	precision	recall	f1-score	support
General reporting	0.94	0.90	0.92	263
Positive	1.00	1.00	1.00	7
Non-conformity	0.85	0.78	0.82	51
Spareparts	0.70	0.91	0.79	34
Safety Issue	0.80	0.67	0.73	6
Warranty	0.84	1.00	0.91	16
Get More Info	0.81	0.85	0.83	60
Quote request	1.00	1.00	1.00	2
accuracy			0.89	439
macro avg	0.87	0.89	0.88	439
weighted avg	0.89	0.89	0.89	439

Figure 6.4: Classification report. Source: Own elaboration

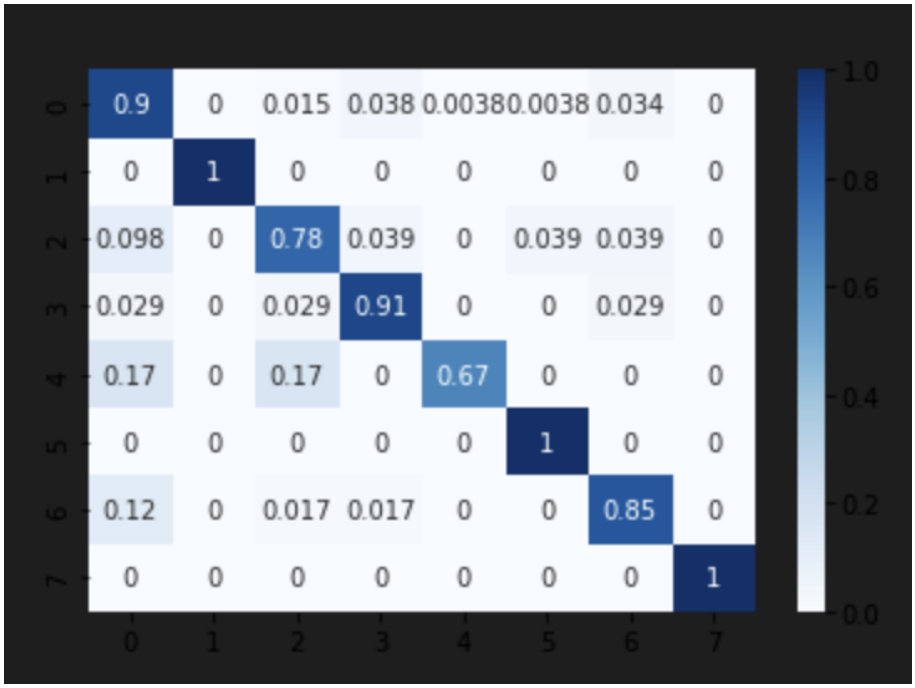

Figure 6.5: Confusion Matrix. Source: Own elaboration

```
Epoch 1 de 5
------------------
Entrenamiento: Loss: 1.1146260826530572, accuracy: 0.6505
Validación: Loss: 0.8411117111508911, accuracy: 0.7659090909090909
Epoch 2 de 5
------------------
Entrenamiento: Loss: 0.5309822654956673, accuracy: 0.849
Validación: Loss: 0.832884935929591, accuracy: 0.8068181818181818
Epoch 3 de 5
------------------
Entrenamiento: Loss: 0.25588434745807165, accuracy: 0.9375
Validación: Loss: 0.763547757335284, accuracy: 0.8522727272727272
Epoch 4 de 5
------------------
Entrenamiento: Loss: 0.12117329346030689, accuracy: 0.9715
Validación: Loss: 0.8408639873897128, accuracy: 0.8454545454545455
```

Figure 6.6: Bert Performance over epochs. Source: Own elaboration

After testing the model on unseen data, it showed a particularly good performance when recognizing **Positive** and **Spareparts** classes.

Regarding BERT, after fine-tuning the model the best performance achieved over the training epochs is shown in the following Figure 6.6. When examining the predictions, it seems that this classifier is particularly good at recognizing **Non-Conformity** cases. After multiple manual reviews of the predictions, it was decided to stick with the BERT model for the upcoming iterations.

CHAPTER 7
Conclusions

This work has been carried out at Kongsberg Maritime company during my internship and it has been proposed to develop a tool to automatically classify service reports to identify cases of interest to the business. The whole project has had a very experimental, trial-and-error character, and what has turned out to work best for the purpose will be summarized below.

In terms of the proposed objectives, the data loading process for the model has been abstracted and made easy to maintain in case of future changes of data sources. After the experimentation with text cleaning, we have opted for regular expressions for text cleaning and *TF-IDF* vectorization for text encoding so that it can be passed to the model. It was decided to establish 8 categories to be identified in the texts: **Safety Issue, Non-Conformity, Warranty, Spareparts, General Reporting, Positive, Quote request, and Get more information**.

Out of all the variables available in the database, the field that has given the best results in terms of predictions has turned out to be **Work Carried Out** and it has also been decided to separate this field into phrases, identify each of these with a double primary key and store it in an auxiliary dataframe, in which the classification will be made at row level, that is, at phrase level.

During the first phase of the project, the one of unsupervised learning, the *Classy Classification* algorithm has been chosen and once it has been put into production and the predictions have been generated, these have been corrected, giving rise to a correctly labeled training set that has allowed entering the second phase of the project, the one of the supervised learning. In this, *Support Vector Machine* model has eventually been chosen as the definitive model and it is being fine-tuned at this moment with subsequent iterations.

In addition, a Custom Rule-based matching system has been successfully designed to identify and graphically mark Kongsberg products in the texts to facilitate reading comprehension when reading a long report.

A form has been developed in Microsoft Access to allow the supervision of the model's performance by the engineers who are the final clients of the tool. The definitive APP is still to be developed and different software options are currently being studied to identify the one that best suits the needs of the company.

A Microsoft Access form has been developed to allow the monitoring of the model performance by the engineers who are the final customers of the tool. The

final APP is still to be developed and different software options are currently being studied to identify the one that best suits the company's needs.

7.1 Constraints Encountered

One of the project obstacles identified is the internal friction against organizational change. There will always be some resistance to change in the organization. This is a natural part of any change process that directly impacts people's daily work. People may feel uncertain about their role or future within the company. Even so, it is possible to mitigate most of the resistance. From day one, clear and concise communication with all stakeholders is key to this mitigation. Worth mentioning are the benefits for individuals, departments, and companies, as well as new personal opportunities that directly or indirectly result from implementation.

Regarding the technical part, the first constraint, and unquestionably the one that has produced less favorable results than anticipated, is the quality of the input data. In this case the spelling issues and the lack of general format of text in the reports made the task much more difficult. No matter how many different preprocessing techniques are applied on the data, if the quality of the data sets is poor, it will be difficult (and frequently directly impossible) to accomplish any data science objectives.

It is obvious that the lack of labeled dataset for the training purposes from the beginning has greatly limited the time dedicated to experimenting with the supervised models, since the majority of the time and effort of the project has been focused on finding a first method unsupervised to get the first predictions.

Another difficulty has been deciding at what level to make the predictions, whether at the level of the entire report or at the level of single sentences. The problem is that many of the sentences need context to be classified correctly, while a complete report may have parts related to more than one class at a time, so the model is not able to decide on just one of them. Nor has a way been found that manages to separate the report by phrases perfectly all the times.

7.2 Future work

As a proposal for future work, a much more extensive experimentation could be carried out in the part of supervised models. As for BERT, more learning rates and other loss functions and optimizers could be experimented with.

Regarding machine-learning models, experiments have been carried out with other models such as XGBOOST [35] and Multinomial Naive Bayes [36] that gave good results but those needed to be refined, which could also be another proposal for future work. It is also necessary to perform additional experiments with other

word embeddings and vectorization techniques, such as GloVe [1] or Word2Vec [2] and with the SMOTE configuration, among others.

It is intuited that as more labeled data is obtained, BERT will end up being the definitive model that will be put into production, so there are infinite things with which to continue the experimentation , starting with learning rates, other loss functions and optimizers or directly other models such as ROBERTa, a model that improves on the masked language modeling objective compared with BERT and leads to better downstream task performance.

Regarding the design of the graphical interface, for now a practical and functional way has been chosen to output the predictions and generate labeled data by getting their corrections, but over time it would be necessary to investigate a software that allows having a beautiful and at the same time functional design which incorporates all the developed functionalities in a way that final users, that is, the life-cycle management engineers are comfortable supervising the performance of the model on a day-to-day basis.

7.3 Relationship between previous studies and present research

The realization of this final degree project has been possible thanks to the knowledge obtained during the last four years of the data science degree. The subjects that have had a special relationship and have helped the most to achieve the objectives of this work are the following:

- The subjects of "Programming" and "Exploratory Data Analysis" for the tasks of exploratory data analysis, data cleaning and data preprocessing.
- Good programming practices, all Python code development, understanding of the source code of certain libraries, strategies to reduce computation time mentioned and code optimization, is linked to the subjects "Programming", "Data Structures and Algorithms" and "Algorithmics".
- All the tasks of text data processing and treatment, deep learning models for sentiment analysis and comprehension of transformer architectures is linked to the "Natural Language Processing" subject.
- The part of supervised learning models, the experimentation carried out with them, their evaluation and deployment are closely related to the subjects "Descriptive and Predictive Models II" and "Scalable Techniques in Machine Learning".
- The understanding of the mathematical basis of the models used and their detailed mathematical formulation in several papers has been possible thanks to the subjects "Mathematical Analysis" and "Continuous Modeling and Simulation".

[1] https://nlp.stanford.edu/projects/glove/

[2] https://datascientest.com/es/nlp-word-embedding-word2vec-es

- Finally, all "Project" subjects have been essential to learn how to apply the theoretical knowledge in a data science project in a practical way, how to organize the different phases of the work and how to structure the content well.

Bibliography

[1] Turing A.M Intelligent machinery, report for National Physical Laboratory, eds. *in Machine Intelligence 7, eds., B. Meltzer and D. Michie*,october, 1950.

[2] Turing A.M Computing machinery and intelligence. *Mind 49*, pp 433-460, 1950.

[3] Turing A.M Can a Machine Think. *The World of Mathematics, ed. James R. Newman*, volume 4, pp 2099-2123, 1956.

[4] Maulud, D., Abdulazeez, A. M. A review on linear regression comprehensive in machine learning *Journal of Applied Science and Technology Trends*, 1(4), 140-147. 2020.

[5] Father's of AI picture last seen: 15 of august of 2022. Consulted in https://medium.com/rla-academy/dartmouth-workshop-the-birthplace-of-ai-34c533afe99

[6] Types of Machine Learning picture last seen: 15 of august of 2022. Consulted in shorturl.at/fW138

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need 12 junio de 2017

[8] Dartmouth Summer Conference last seen: 14 of august of 2022. Consulted in https://250.dartmouth.edu/highlights/artificial-intelligence-ai-coined-dartmouth

[9] Precision and Recall comparison picture last seen: 1 of august of 2022. Consulted in https://en.wikipedia.org/wiki/Precision_and_recall

[10] Confusion matrix picture last seen: 7 of august of 2022. Consulted in https://www.researchgate.net/figure/Confusion-Matrix-for-Binary-Classification-7_fig1_350487701

[11] Neuronal network structure last seen: 27 of august of 2022. Consulted in https://www.ibm.com/cloud/learn/neural-networks

[12] ReLU Activation Function last seen: 31 of august of 2022. Consulted in https://iq.opengenus.org/relu-activation/

[13] J. Schmidhuber. Deep Learning in Neural Networks: An Overview *Neural networks*, 62, 86–117, 2013

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun Deep residual learning for image recognition *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016

[15] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean Google's neural machine translation system: Bridging the gap between human and machine translation *CoRR*, abs/1609.08144, 2016.

[16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint*, 2016.

[17] Xenonstack vlog last seen: 20 of august of 2022. Consulted in https://www.xenonstack.com/blog/log-analytics-deep-machine-learning

[18] Dominik Stammbach and Elliott Ash DocSCAN: Unsupervised Text Classification via Learning from Neighbors *CoRR*, vol. abs/2105.04024, 2021.

[19] Medium NLP pipeline last seen: 25 of august of 2022. Consulted in https://medium.com/predict/how-does-nlp-pre-processing-actually-work-8d097c179af1

[20] Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool Scan: Learning to classify images without labels. *In Proceedings of the European Conference on Computer Vision*, 2020.

[21] Youngjoong Ko and Jungyun Seo. Automatic Text Categorization by Unsupervised Learning. *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, pages 453-459, 2000.

[22] Few-Shot Learning Guide last seen: 26 of august of 2022. Consulted in https://www.v7labs.com/blog/few-shot-learning-guide

[23] Towardsdatascience: Computer vision last seen: 26 of august of 2022. Consulted in shorturl.at/BCFHK

[24] Pandora Intelligence last seen: 28 of august of 2022. Consulted in https://www.pandoraintelligence.com/

[25] Classy Classification repository last seen: 30 of august of 2022. Consulted in https://github.com/Pandora-Intelligence/classy-classification

[26] Rasa NLU last seen: 30 of august of 2022. Consulted in https://github.com/RasaHQ/rasa

[27] Support Vector Machine last seen: 31 of august of 2022. Consulted in shorturl.at/d0368

[28] TF-IDF simplified last seen: 31 of august of 2022. Consulted in https://towardsdatascience.com/tf-idf-simplified-aba19d5f5530

[29] SMOTE Oversampling last seen: 1 of september of 2022. Consulted in https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[30] Torrey, L., Shavlik, J. Transfer learning *In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, (pp. 242-264), IGI global,2010.

[31] Yinhan Liu and Myle Ott and Naman Goyal and Jingfei Du and Mandar Joshi and Danqi Chen and Omer Levy and Mike Lewis and Luke Zettlemoyer and Veselin Stoyanov RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, vol. abs/1907.11692, 2019.

[32] Blei, D. M., Lafferty, J. D. A correlated topic model of science. *The annals of applied statistics*, 1(1), 17-35, 2007

[33] BERTopic Last seen: 30 of august of 2022. Consulted in https://towardsdatascience.com/implement-your-topic-modeling-using-the-bertopic-library-d6708baa78fe

[34] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805v2*, October 2018

[35] Mustika, W. F., Murfi, H., Widyaningsih, Y. Analysis accuracy of xgboost model for multiclass classification-a case study of applicant level risk prediction for life insurance. *In 2019 5th International Conference on Science in Information Technology*, (ICSITech) (pp. 71-77). (2019, October)

[36] Kibriya, A. M., Frank, E., Pfahringer, B., Holmes, G. Multinomial naive bayes for text categorization revisited. *In Australasian Joint Conference on Artificial Intelligence* (pp. 488-499). Springer, Berlin, Heidelberg, 2004, December

APPENDIX A
Factory Method Design

One of *Good practices* that I have learned and applied in this project is the Factory Pattern. Factory Method is a creational design pattern used to create concrete implementations of a common interface. It divides the creation of an object's interface from the code that uses that interface. For instance, an application needs an object with a particular interface in order to function. Some parameter identifies the actual implementation of the interface. Each method of the class will have a single, well-defined responsibility. A common interface in this case is the one that reads the data from a specified source and returns a pandas dataframe. So separate implementations for each logical path are provided: one for reading the data from SQL Database and another one for reading the data from CSV files. Lastly a separate component is created in order to decide which concrete implementation should be used based on the parameter specified: string that specifies a format to use.

This way the refactoring of the code is made and its internal structure is improved, so in the future we can easily add new types of data sources without disturbing the existing client code.

Below is detailed briefly the concrete implementation of this design in this project. So basically, there is a Abstract Base Class (which inherits from ABC object from *ABC* library) which determines a basic representation of a DataLoader. The abstract classes within it state that each DataLoader should have at least these two methods: the first one in charge of reading a necessary configuration from a given json file (with path information and headers formatting) and the second one is the responsable of creating a pandas datarame taking into account the configuration given in the json file.

```
1  class DataLoader(ABC):
2      """Basic representation of a DataLoader"""
3
4      @abstractmethod
5      def load_json(self, path: pathlib.Path):
6          """Loads the neccesary config from the necessary json"""
7
8      @abstractmethod
9      def create_df(self, config):
10         """Creates a dataframe with the config from json"""
```

Listing A.1: Abstract Base Class definition

Then two classes are created by inheriting from the previous Abstract Base Class, the csv file loader and the excel file loader. Each of them have defined these two methods as specified above and the result of applying them both is a unified pandas dataframe.

```
class csv_loader(DataLoader):
    """Loads csv files"""
    def __init__(self):
        self.path_to_json_config = './config/config_csv.json'

    def load_json(self):
        with open(self.path_to_json_config, "r", encoding="utf-8") as f
            :
            json = json.load(f)
        return json


    def create_df(self)-> pd.DataFrame:
        json = self.load_json()
        df = pd.read_csv(json["ipdb"]['file_path']['source_dir'] + json
            ["ipdb"]['file_path']['file_name'],  sep = json["ipdb"]['
            config']['sep'], encoding = json["ipdb"]['config']['
            encoding'] , skiprows=json["ipdb"]['config']['skip_rows'])
        columns_input = json["ipdb"]['columns']['column_names_input']
        columns_output = json["ipdb"]['columns']['column_names_output']
        for (previous_col_name, new_col_name ) in zip(columns_input,
            columns_output):
            df.rename(columns={previous_col_name: new_col_name},
                inplace = True)
        return df
```

Listing A.2: Example CSV Loader

Finally a separate component is needed in order to decide which implementation to use. It receives a string indicating from which source is the data coming from and based on this it returns a class object defined for treating this kind of data. Lastly a method from this class in charge of creating the dataframe is called and a dataframe is created and returned.

```
def load_dataframe(format:str):
    factories = {
        "csv": csv_loader(),
        "excel": excel_loader()
    }

    if format in factories.keys():
        return factories[format].create_df()
    else:
        print(f"Unknown format: {format}.")
```

Listing A.3: Separate component

APPENDIX B
BERT transfer learning and fine-tuning

```
from transformers import BertModel, BertTokenizer, AdamW,
    get_linear_schedule_with_warmup
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder

# Initiation of params.
RANDOM_SEED = 58
MAX_LEN = 200 # the average sentence length
BATCH_SIZE = 6
NCLASSES = 8

np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

# Import the model and its own tokenizer
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)


# The creation of pytorch dataset

class MyDataset(Dataset):

  def __init__(self,sent,label,tokenizer,max_len):
    self.sent = sent
    self.label = label
    self.tokenizer = tokenizer
    self.max_len = max_len

  def __len__(self): #pytorch always requires this method
      return len(self.sent)

  def __getitem__(self, item): #the method that pytorch will be calling
      to create the batches of size 6 (BATCH_SIZE)
    sent = str(self.sent[item])
    label = self.label[item]
    encoding = tokenizer.encode_plus(
```

```
        sent,
        max_length = self.max_len,
        truncation = True, #if there are more than MAX_LEN tokens it
            eliminates the rest
        add_special_tokens = True, # to add CLS and SEP token
        return_token_type_ids = False,
        padding='max_length',
        return_attention_mask = True, #encodes as 1 that part of
            sentence which is taken into account (cls token, words
            token) during the training and encodes as 0 the rest (
            padding tokens )
        return_tensors = 'pt'
        ) # encoding.keys() =['input_ids', 'attention_mask']


    return {
          'sent': sent,
          'input_ids': encoding['input_ids'].flatten(),
          'attention_mask': encoding['attention_mask'].flatten(),
          'label': torch.tensor(label, dtype=torch.long)
      }



# Data loader method that uses the DATASET class we defined before in
    order to create batches of data and pass them to the model
    afterwards:

def data_loader(df, tokenizer, max_len, batch_size): #the function that
     returns the data split into batches
  dataset = MyDataset(
      sent = df.sent.to_numpy(),
      label= df.class1.to_numpy(),
      tokenizer = tokenizer,
      max_len = MAX_LEN
  )

  return DataLoader(dataset, batch_size = BATCH_SIZE, num_workers = 2)

# Split the data into train/test partitions
train, test = train_test_split(df, test_size=0.18, random_state = 58)
train_data_loader = data_loader(train, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = data_loader(test, tokenizer, MAX_LEN, BATCH_SIZE)

# Definition of the model

class BERTSentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(BERTSentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
    self.drop = nn.Dropout(p=0.35)
    self.linear = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    _, cls_output = self.bert(
        return_dict=False,
        input_ids = input_ids,
        attention_mask = attention_mask
```

```
92      ) #get the ouput formed by cls tokens out of BERT layer
93      drop_output = self.drop(cls_output) #pass the output of BERT
            through the dropout layer
94      output = self.linear(drop_output) # pass the ouput of the dropout
            layer to the neuronal network layer
95      return output
96
97  model = BERTSentimentClassifier(NCLASSES)
98  model = model.to(device) #indicate that we want to train the model
        using the set device: using the GPU
99
100 # Training process
101 # For a typical Pytorch training cycle, we need to implement the loop
        for epochs, iterate through the mini-batches,
102 # perform feedforward pass for each mini-batch, compute the loss,
103 # perform backpropagation for each batch and then finally update the
        gradients.)
104
105 EPOCHS = 5 #training iterations
106 optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
107 total_steps = len(train_data_loader) * EPOCHS #len(train_data_loader)
        equals the batch size (6)
108 scheduler = get_linear_schedule_with_warmup(
109     optimizer,
110     num_warmup_steps = 0,
111     num_training_steps = total_steps
112 ) #the scheduler aims to reduce the lr through the following iterations
        , this intents to optimize the training process
113
114
115
116 def labels_to_class_weights(labels, nc=80):
117      # Get class weights (inverse frequency) from training labels
118      labels = np.concatenate(labels, 0)  # labels.shape = (866643, 5)
            for COCO
119      classes = labels[:, 0].astype(np.int)  # labels = [class xywh]
120      weights = np.bincount(classes, minlength=nc)  # occurences per
            class
121      weights[weights == 0] = 1  # replace empty bins with 1
122      weights = 1 / weights  # number of targets per class
123      weights /= weights.sum()  # normalize
124      return torch.Tensor(weights)
125
126 loss_fn = nn.CrossEntropyLoss().to(device) # the error function that
        will be minimized throught the iterations
127
128
129 # Methods that make the training process iterable
130 def train_model(model, data_loader, loss_fn, optimizer, device,
        scheduler, n_examples):
131   model = model.train()
132   losses = []
133   correct_predictions = 0
134   for batch in data_loader:
135     input_ids = batch['input_ids'].to(device)
136     attention_mask = batch['attention_mask'].to(device)
137     labels = batch['label'].to(device)
138     outputs = model(input_ids = input_ids, attention_mask =
            attention_mask)
```

```
139     _, preds = torch.max(outputs, dim=1) #pytorch selects the max
            weighted class as a prediction
140     loss = loss_fn(outputs, labels) # the way of calculating manually
            the loss
141     correct_predictions += torch.sum(preds == labels) # we check how
            many of predictons are right
142     losses.append(loss.item())
143     loss.backward() # we feed the model with this calculated loss
            backwardly and update the weights
144     nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0) #this
            prevents the gradient from growing too fast normalizing it to
            scale -1 to 1
145     optimizer.step() #step which updates the weights
146     scheduler.step() #step which updates the lr
147     optimizer.zero_grad() #we retart the gradients for the following
            iteration
148   return correct_predictions.double()/n_examples, np.mean(losses)
149
150 def eval_model(model, data_loader, loss_fn, device, n_examples):
151   model = model.eval() # this allows to 'block the model' from updating
          the weights, it puts it on the evaluation mode
152   losses = []
153   correct_predictions = 0
154   with torch.no_grad(): # this indicates that no weight should be
          updated
155     for batch in data_loader:
156       #same steps as in training method
157       input_ids = batch['input_ids'].to(device)
158       attention_mask = batch['attention_mask'].to(device)
159       labels = batch['label'].to(device)
160       outputs = model(input_ids = input_ids, attention_mask =
              attention_mask)
161       _, preds = torch.max(outputs, dim=1)
162       loss = loss_fn(outputs, labels)
163       correct_predictions += torch.sum(preds == labels)
164       losses.append(loss.item())
165   return correct_predictions.double()/n_examples, np.mean(losses)
166
167 # Training
168 for epoch in range(4):
169   print('Epoch {} de {}'.format(epoch+1, EPOCHS))
170   print('------------------')
171
172   train_acc, train_loss = train_model(model, train_data_loader, loss_fn
          , optimizer, device, scheduler, len(train))
173
174   test_acc, test_loss = eval_model(model, test_data_loader, loss_fn,
          device, len(test))
175
176   print('Entrenamiento: Loss: {}, accuracy: {}'.format(train_loss,
          train_acc))
177   print('Validaci n: Loss: {}, accuracy: {}'.format(test_loss,
          test_acc))
```

Listing B.1: BERT

APPENDIX C
Objetivos del desarrollo sostenible (ODS)

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este proyecto guarda relación con varios de los Objetivos de Desarrollo Sostenible de la Organización de las Naciones Unidas. Los dos objetivos con los que hay una mayor relación son el objetivo 8, 'Trabajo decente y crecimiento económico' y el objetivo 9, 'Industria, innovación e infraestructuras'. También se puede relacionar con los objetivos 13, 'Acción por el clima' y 14, 'Vida submarina' de manera menos estrecha.

Esto se debe a que en este proyecto se ha desarrollado un nuevo software para un negocio que afectará de forma positiva al crecimiento económico y el uso optimizado de los recursos necesarios para realizarlo, de ahí la relación con los objetivos número 8 y 9.

Por último, al contribuir este proyecto a la automatización de procesos en un negocio tan grande (y en un futuro tal vez en otros negocios del mismo sector) tiene especial correspondencia con el objetivo 13, Acción por el clima. Al ser Kongsberg una empresa del sector Marítimo, tiene mucha responsabilidad de cuidar y mantener la vida submarina. Si esta herramienta detecta correctamente las piezas que se mencionan como defectuosas en los reportes, contribuirá a reducir el número de componentes defectuosos instalados en los buques que comprometen el bienestar del medio ambiente y el bienestar submarino.