FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation: **Applied Data Science**

The spring semester, **2022**

Open / Confidential

Author: **Tsegazab Tesfay**

.....
(signature author)

Course coordinator:

Supervisor(s): **Ferhat Özgur Catak**

Thesis title:

Using various Natural Language Processing Techniques to Automate Information Retrieval

Credits (ECTS): 30

Keywords:

Natural Language Processing,
Machine Learning,
Information Extraction

Pages: **76**

+ appendix: **GitHub-link**

Stavanger, **14/June/2022**
date/year

Approved by the Dean 30 Sep 21
Faculty of Science and Technology

Using various Natural Language Processing Techniques to Automate Information Retrieval

Tsegazab Tesfay

14 June 2022

Preface

This thesis is a feasibility study of an information extraction technique based on Natural Language Processing that employs various machine learning and deep learning algorithms. This project was written as part of the Master of Science degree program in Applied Data Science at the University of Stavanger.

This project was assigned by Autility AS in response to my request for masters thesis project. The request sparked a lively discussion with Autility AS's CEO, Sebastian Videhult. The discussion about Autility's existing projects was then continued with Autility AS's CTO, André Keane. I've had numerous and continuous meetings, specifically with André Keane, regarding project clarification, advice, and gathering as much quality data as possible. Autility, the University of Stavanger, and I agreed right before the semester started that the project would be about Natural Language Processing, which uses various techniques to retrieve information. This served as the foundation for the project.

I would like to thank my supervisor, Professor Ferhat Özgur Catak, for his helpful feedback and assistance. Every time I attend his meeting, I am inspired and encouraged. I'd also like to thank Sebastian Videhult, CEO of Autility, for the opportunity and for introducing me to the company's vision on my first day. Furthermore, I would like to thank Autility's CTO, André Keane, for introducing me to the company on a deeper level. André explained not only the significance of this project, but also the overall concept of Autility AS's existence. In general, I can't thank him enough for the explicit information I received, which could lead to significant progress for our society and, ultimately, our world on the topic of climate change in the coming years. Finally, I'd like to express my gratitude to my parents and siblings for always loving and supporting me. Without you, I couldn't have done it! Last but not least, I'd like to thank all of my friends for their advice and help.

This thesis is intended for those who want a thorough introduction to the concept of machine learning and deep learning through the use of multiple algorithms for classification and Natural Language Processing techniques for information extraction.

Executive Summary

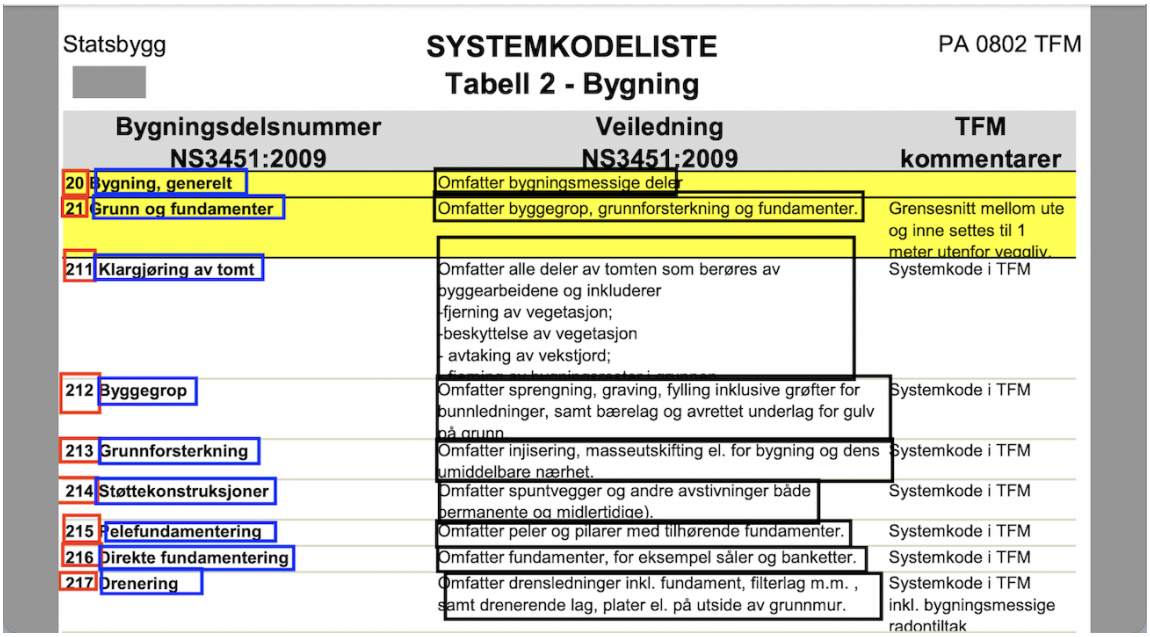
The existence of *Natural Language Processing*(NLP) provides numerous benefits, including the understanding and analysis of unstructured data, as well as the efficient and precise automation of real-time processes. Despite the fact that NLP began in the 1940s, the importance of having an application that uses the benefits of NLP has never been greater than in the last two decades. This is because as the number of people who have access to the internet or digital devices grows, so does the size of the data collected. Thus, NLP and automated processes play a significant role in the quality and performance of services that users encounter.

Datasets are not always structured or automated. This is due to the size of the data or the companies' age in terms of data collection. Several studies have shown that unstructured data contains useful information that, when managed properly, can point businesses in the right direction. To address these issues, it is critical to combine NLP and *Machine Learning*(ML) or *Deep Learning*(DL) algorithms. In other words, algorithms can deal with structured, unstructured, or both types of data. The algorithms' contributions are to automatically learn the language pattern in the given text and use that pattern to identify the unseen or validation data. Hyperparameter optimization are also performed in both supervised and unsupervised type of machine learning to make the algorithms as flexible as possible while achieving the desired results.

The goal of this thesis is to develop an automated system that classifies files using various NLP in conjunction with the ML/DL algorithm that produces the best performance results. Autiliy AS is a young company focused on digitalization buildings. There are thousands of structured and unstructured files in Autility. Autility intends to use an automated system to extract information and classify files based on the system-code labeled "SYSTEMKODELIST NS3451". The "SYSTEMKODELISTE NS3451" is the *"backbone"* for the entire system creation process. The first part of the main "SYSTEMKODELISTE NS3451" from Norwegian Statsbygg is shown in figure 1. Only 12 rows of the standard "SYSTEMKODELISTE NS3451" are displayed. Its full version is included as an attachment. The red square in the figure represents the *building part number*, which will be referred to as *system-code* in this project, the blue square represents the name of the *building part*, and the black square represents *guidance* for the given building part. Chapter 3 contains a brief explanation of how it is integrated into this project.

The labeled dataset produces models with an average accuracy of roughly 85%. However, because the dataset contains far more unstructured files than structured files, research into algorithms that handle both structured and unstructured data is critical. Because many of the files contained drawings of buildings and pictures, the results of semi-supervised algorithms indicated the importance of formal language. To ensure consistent performance

and a system with less overfitting, textaugmentation and hypertunneling are used. The assumptions made and the challenges faced are documented throughout this project. A few algorithms are presented in detail, along with their theoretical and mathematical concepts.



Statsbygg

SYSTEMKODELISTE
Tabell 2 - Bygning

PA 0802 TFM

Bygningsdelsnummer NS3451:2009	Veiledning NS3451:2009	TFM kommentarer
20 Bygning, generelt	Omfatter bygningsmessige deler	
21 Grunn og fundamenter	Omfatter byggegrop, grunnforsterkning og fundamenter.	Grensesnitt mellom ute og inne settes til 1 meter utenfor veggliv.
211 Klargjøring av tomt	Omfatter alle deler av tomten som berøres av byggearbeidene og inkluderer -fjerning av vegetasjon; -beskyttelse av vegetasjon - avtaking av vekstjord;	Systemkode i TFM
212 Byggegrop	Omfatter sprengning, graving, fylling inklusive grøfter for bunnledninger, samt bærelag og avrettet underlag for gulv på grunn	Systemkode i TFM
213 Grunnforsterkning	Omfatter injisering, masseutskifting el. for bygning og dens umiddelbare nærhet.	Systemkode i TFM
214 Støttekonstruksjoner	Omfatter spuntvegger og andre avstivninger både permanente og midlertidige).	Systemkode i TFM
215 Pelefundamentering	Omfatter peler og pilarer med tilhørende fundamenter.	Systemkode i TFM
216 Direkte fundamentering	Omfatter fundamenter, for eksempel såler og banketter.	Systemkode i TFM
217 Drenering	Omfatter drensledninger inkl. fundament, filterlag m.m. , samt drenerende lag, plater el. på utside av grunnmur.	Systemkode i TFM inkl. bygningsmessige radontiltak

Figure 1: Classification structure

After this thesis, the system implemented and demonstrated here will be used in a practical case with continuous improvement.

Contents

Chapter 1

Introduction

This chapter introduces the thesis's quick overview, the problem that will be solved in this project, and the thesis's structure.

1.1 Background

Nowadays, data is collected from almost every daily activity. To name a few, shopping, web-commerce (where a lot of transactions take place online), and so on. And all of this activity results in a large amount of data being collected from billions of people every day. Furthermore, computers are becoming less expensive and more powerful, allowing people to communicate more easily while also allowing powerful corporations to analyze data. These businesses become more competitive in terms of providing better customer service and attracting new customers. When we consider the importance of data from a scientific standpoint, we can use it to identify new planets and analyze changes in outer space. Data can take the form of images, videos, text documents, and so on. Most of the data we would deal with in NLP are collections of data with some attributes, also known as features or variables. The attributes may define the property or characteristic of the objects in the data collection. A person's eye color and temperature are examples of properties or characteristics.

In this project, data is gathered from real estate owners who apply for assistance from Autility AS, which is then handled by Autility members. Manually classifying the applications as structured or unstructured is required. These applications are classified using the "SYSTEMKODELIST NS3451" code from the Norwegian Directorate of Public Construction and Property (which can be found in the attachment). The code is an identification number that provides detailed information about the application's location, system, component, and type. This process takes time, and due to the large number of applications, much

information may be lost in the future. Autility AS wants to develop a system that can assist advisers in automating the task in order to avoid these and use the data for a variety of other purposes within automating, which is the main goal for Autility. In other words, given an application, the system should both classify it using "SYSTEMKODELIST NS3451" and extract information, such as the names of organizations, dates, and locations.

To achieve a consistent result, the process of developing this type of system includes NLP, ML, DL, and Artificial Intelligence (AI) as a whole. And all of these main topics, as well as other important topics like structured and unstructured data types, will be covered in the Theory section. There is also a brief theoretical explanation in several NLP and general information extraction (IE) sub topics. This thesis is analogous to a research process in which I examine which ML models and information in the application are most reliable for producing the output - system-code.

1.2 Autility

Autility is a company that assists real estate owners, property managers, and tenants in optimizing the use of their building portfolios for maximum profit. In order to accomplish this, Autility makes use of cutting-edge technologies such as Digital Twins, which give customers real-time access to valuable objects; AI, which is comprised of a number of different methods (Computer Vision, Anomaly Detection, Predictive Analytics), Internet of Things, Automation and so on. In other words, the primary reason why Autility exists is to digitalize buildings while also making certain that they are prepared for the future. In order to achieve this goal, Autility frees data from silos, ensures that systems are able to communicate with one another, and employs AI in order to optimize and simplify administration, operation, and tenant relations. Autility takes the first step toward achieving this objective by connecting buildings and systems to the main platform. On this platform, AI works to optimize and simplify operations, and information is presented in a manner that is understandable and customized for each of their solutions.

After connecting the buildings and the data associated with them, Autility will be able to begin processing the data in the platform. This will involve using AI to automate tasks and provide decision support to users before the visual user interfaces are presented to them.

1.3 Objectives and Problem Description

This thesis provides a feasibility analysis of the viability of applying current technology for natural language processing. Its primary purpose is to provide Autility AS, a system that

can both categorise system code and extract information using multiple natural language processing techniques in conjunction with machine learning. This paper presents:

- Background information.
- Implementation of Use Case.
- Experiments and Results.
- An evaluation of the obtained system's quality.

1.4 Approach

The ability to deal with both structured and unstructured data types using a variety of NLP techniques without having in-depth knowledge of the domain is one of the primary focuses of this thesis. Another approach is to become familiar with the basic mathematics that lies at the heart of each of the ML and DL algorithms that will be utilized in this undertaking. The verification of the result through the application of the principles of cross validation and greedsearch can be seen as an approach in addition to the approaches of the models that were compared to one another.

1.5 Limitation

The benefits of NLP may differ depending on what we are capable of with today's technology. The long-term goal is to create machines that can understand what we humans say and interact with us in a human-to-human manner. Finding out how the brain converts language into action after understanding it is another task that is expected to be solved in the future. Currently, NLP can be used for language understanding, machine translation (though there are limitations in some cases), sentimental analysis, paraphrasing, and so on. Even though there has been significant progress in the field of NLP, there are still limitations. For example, when someone speaks on the radio, a visual context is lost. If a person says "I want that", the interpretation of the sentence may be lost both from NLP prespective and from humans prespectives due to a lack of visual scene, i.e one can not understand what "that" refer to. The project's results and implementation are limited by the quality of data and memory capacity.

1.6 Arrangement of this Thesis

The thesis process is divided into several main and sub-tasks. Natural Language Processing and ML are briefly discussed in Chapter 2. Following a brief explanation of the theoretical and mathematical concepts underlying ML and DL algorithms, an explanation of model evaluation with examples is provided. Chapter 3 describes the thesis's main problem and compares all of the algorithms shown in Chapter 2. The chapter begins with data exploration, then data preparation, and finally text vectorization for further operation by the algorithms. This chapter also includes the algorithm results. Chapter 4 discusses the results from Chapter 3 as well as the assumptions used and the challenges encountered during the thesis process. In this Chapter it will be included the environment accounts together with the conclusion reached based on the outcome.

Chapter 2

Theory

2.1 Natural Language Processing

Humans have always been able to communicate with one another through the use of a common language. And, despite differences or errors in the form of mispronunciations, accent changes, and other difficulties, we can still understand each other. This is what gives humans and human language such power [1]. When it comes to computers, one could argue that they have always been capable of understanding their own language known as *code*. However, because humans must manually write all of this code, the language is "extremely limited" in vocabulary and restricted. For example, if a programmer makes a minor spelling or syntax error, the program will crash and the problem will remain unresolved.

Another example that can clarify a point of limitation that may occur in our daily lives is where a group of doctors attempting to identify a source of infectious diseases and warning the population. To do so, doctors want to first figure out why and under what conditions people get the diseases by analyzing previous cases as well as cases from the internet that show similar symptoms. So they begin to examine their data, which includes pictures, notes, journals, and so on. The group of doctors can be overwhelmed by the amount of data because going through and making sense would take time. And the computer doesn't know how to decide this because it can't understand the context that words are in. After all, understanding a word boils down to understanding the words around it, because the word itself has little meaning; it all depends on the context. As a result, the computer is unable to make sense of the data and, as a result, cannot truly assist the doctors. However, given how powerful computers are in terms of memory, speed, and intelligence, computers would be far more useful if they could understand and process our language rather than only incomprehensible 1s and 0s. NLP is the solution to the problem. Doctors can analyze unstructured data in minutes using NLP.

Siri and Alexa, for example, can understand our words and context clues to improve our lives. Google can not only predict results as we type in the search bar, but it can also examine the big picture and recognize more than just the words we type. Our phones and computers can anticipate what we will type, complete our sentences, and suggest appropriate words. By identifying clues and patterns in e-mails and written reports, NLP can solve larger problems such as crimes and even diseases. So it is clear that NLP is extremely powerful and, when used correctly, has the potential to significantly improve our lives. Every day, we each say hundreds of sentences and communicate with a wide range of people. We know that a sentence is ultimately composed of a noun phrase and a verb phrase, and we also know that techniques such as part-of-speech tagging and chunking can be used. These parts of speech give NLP the ability to understand context. The advancement of NLP can be seen by comparing the ancient computer where word confusion occurs due to similar words in a sentence. In other words, NLP can figure out or understand the context - see what phrase a word belongs to - by analyzing hundreds of sentences and different word patterns. The computer can then identify the main subject of each sentence. Other techniques that can be broadly classified into two categories, *syntax* and *semantics*, will be discussed in subsequent sections.

NLP is a branch of AI that is used in this thesis to extract information from textual operations. When performing any type of analysis, we frequently have a large number of numerical values at our disposal, such as sales figures, physical measurements, and quantifiable categories. And computers are very good at dealing with direct numerical data. However, when it comes to text data, we as humans can easily tell what information is stored in a text. The text form could be a PDF file, an email, a book, or any of the many others to which we have access. Again, humans understand how to read natural language, but computers require specialized processing techniques to comprehend raw text data. One of the difficulties may be that the text data is highly unstructured and may be in multiple languages. To address this, NLP attempts to use a variety of techniques to create structure from raw text data. And some of the most important tools in this project are built-in libraries like Spacy [1] and NLTK[2]. These tools will be discussed further in the following sections. Natural Language Processing is commonly used to extract information from text, emails, and classify it as spam versus a legitimate email, or to classify texts as positive or negative. In order to accomplish this, some classifiers must be hired. As a result, a few ML algorithms should be presented to classify and return quality results.

[1] https://spacy.io/models/nb
[2] https://www.nltk.org/

2.2 Information Extraction

Information Extraction (IE) is the process of extracting actual organized information from unstructured input [8].This entails providing important elements of unstructured data in a machine-readable way. IE is very useful for many commercial applications such as Business Intelligence, automatic annotation of web pages, text mining, and knowledge management [5]. Various subtasks involved in IE are: Named Entity Recognition(NER), Named Entity Linking (NEL), Coreference Resolution(CR), Temporal Information Extraction, Relation Extraction (RE), Knowledge Base Construction and Reasoning [3].

The extraction of information occurs at the beginning of the working process with raw data. That is, the information extraction method is calculated at the preprocessing level and the efficiency of different Information Extraction activities is dependent on that. Tokenization, stemming, and lemmtazation which are main part of preprocessing, will be discussed and shown in the implementation part. In order to tag each word from several POS classes, such as noun, raw data sentence segmentation and tokenization are computed. And then ”named entity recognizer” assigns a specific named entity class from multiple classes, such as person, organization, place, date, time, money, percent, e-mail address, and web-address.

Named Entity Recognition is a word-level tagging issue in which each word in a phrase is assigned a named entity tag. Features such as affix, capitalization, punctuation, output of syntactic analyzers (i.e. POS taggers, chunkers), and external resources such as gazetteers, word embeddings, and word cluster ids are fed to the classifier, and the output is the labeled tag in the form of person (PER), organization (ORG), location (GPE), etc [10].

2.3 Machine Learning

ML is a branch of AI [9] and it is said that - **“The field of study that gives computers the ability to learn without being explicitly programmed”** - by Arthur Samuel. This tells that ML is the process of discovering algorithms based on what is learned using data. Another good definition of ML that is explain more explicitly is by Tom M. Michell, 1997 and it stats that - **“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P, improves with experience E”**[4].

The number of times a process has been completed is represented by the experience E, and the process can be supervised, unsupervised, or semi-supervised. While “tasks T” are tasks that can be found in both the predictive and descriptive methods. The predictive

[3] https://arxiv.org/pdf/1807.02383.pdf

[4] https://www.cin.ufpe.br/~cavmj/Machine%20-%20Learning%20-%20Tom%20Mitchell.pdf

method employs some attributes to forecast the value of another attribute, and the task may include classification, regression, time series analysis, and so on. When it comes to descriptive data, it is about determining what happened in the past data by analyzing the stored data; the task can be clustering, summarizing, association rule, and so on. And "performance measure P" is about model evaluation in general, providing answers on how well a given model performs, among other things. The "Model evaluation" subsection refers to an explicit explanation of model evaluation.

There are numerous definitions of ML that go beyond the two mentioned above. Other authors who have defined ML including Stanford, McKinsey and Co. However, according to all these authors, there are three types of ML methods, i.e. a group of specific algorithms that solve a specific problem. These techniques include supervised (predictive), unsupervised (descriptive), and reinforcement learning. The emphasis in this paper will be solely on supervised ML, unsupervised ML, and a hybrid of the two known as semi-supervised ML.

Several ML algorithms are used in this project to classify manually structured files to their correct system-code in "SYSTEMKODELISTE NS3451" from statsbygg. Artificial Neural Networks, Naive Bayes Classifiers, Support Vector Machines, Decision Trees, Random Forest Classifier, Nearest-Neighbor Classifiers, K-Means, and Expectation Maximization are some of the algorithms included in this thesis.

2.3.1 Supervised Learning

When performing supervised learning, the outcome of the data being input is referred to as the label or target[2]. The relationship between the data that was input, which is also referred to as the record, and the effects that led to the result was the primary problem that needed to be solved. Even though there are a lot of different ways to find relationships, the one that is used the most frequently is to first train on a portion of the data that has been labeled (usually around 70%), and then use the other portion of the data to test or validate the findings. The model may then be retained if it satisfies the requirements for the issue that is to be solved, such as having a high level of performance, being efficient, and so on. If the performance of the model is unsatisfactory, model building iterations can be carried out until the required level of performance on test or validation data is achieved. This procedure is illustrated in great detail in the section of the project devoted to its implementation. It is important to note that supervised ML can take the form of either classification or regression, the latter of which is the primary focus of this thesis. CART, which stands for "Classification and Regression Tree" is another name for this type of tree.

Since this kind of ML is frequently used for automatic detection, both Autility and I have expressed a desire to use it to classify each file in the documents. This desire stems

from the fact that this type of ML was first introduced at the beginning of this thesis. That is to say, it is essential to train a ML model with a significant quantity of data that has been labeled. For instance, we might present a model with thousands of pictures of dogs and cats, each of which would have a label indicating whether or not the picture depicted a dog or a cat. In addition, the model will continually try to guess the content of the images and will continue to improve based on whether its guesses are correct or incorrect. It is important to note that the term "supervised" can be used to refer to either classification or regression, but here the main focus is on classification. In other word, every file used to train the model must be marked with a system code from which the model may learn. The dataset retrieved from Autility, each have their own distinct format for the available labels. When the standard system code "SYSTEMKODELISTE NS3451" and the title of the system code retrieved from the folder name are compared, they are discovered to have the identical meaning. Artificial Neural Networks, Naive Bayes Classifiers, Support Vector Machines, Decision Trees, and Nearest-Neighbor Classifiers are some of the most well-known classifier algorithms utilized in the supervised of type ML. Each of these algorithms has its own subsection that talks into further detail.

2.3.2 Unsupervised Learning

Unsupervised learning, as opposed to supervised learning, is used on data that does not have a label. Datasets without labels are common, and manually labeling them is often expensive in terms of both money and time. The problem that unsupervised learning attempts to solve is identifying the pattern of the given data and grouping them again where the attributes are of the same kind.

This project's dataset contains a significant amount of unlabeled data. The number of these files is specified in chapter three. The original data was also subjected to clustering using the k-means algorithm. That is, the data was collected prior to the text augmentation, which was also briefly explained in chapter three. The clustering is done to see if there was a pattern that led to the data being grouped. The number of files with manually labeled from the standard "SYSTEMKODELISTE NS3451" in the dataset that will be used in this project is significantly less than the number of files without system code. This is demonstrated in the *Exploratory Data Analysis*(EDA) section of Chapter 3. Because the ratio of labeled to unlabeled data is so diverse, investigating the semi-supervised part of ML is an alternative to obtaining a large amount of correctly labeled data at the end of the project.

2.3.3 Semi-Supervised

The supervised and unsupervised learning approaches are combined in the semi-supervised learning method. When both labeled and unlabeled data are presented, and the size of the unlabeled data is typically much larger than the size of the labeled data, this method is preferred. In this thesis, it is clarified that having access to a variety of different kinds of semi-supervised algorithms is a significant advantage. This is due to the fact that the end result will be of higher quality.

- EM
- Generative adversarial networks(GANs)
- Semi-supervised support vector machines(S3VMs)
- graph based methods
- Markov Chain method

These are some of the most well-known algorithms for dealing with problems in the semi-supervised ML domain. However, given to time limitations and data quality, the EM picked among the others.

2.3.4 Classification

Label categorization is considered to be one of the most important aspects of this project (system code). For the purpose of assigning the text file to the system code, ML algorithms are required. Finding the appropriate algorithm that correctly categorizes the texts based on the criteria described in the following sections is necessary to make sure that the quality of this project will be maintained. To put it another way, it is fundamental to have a classification model that can determine the user's system code based on the texts that are provided.

The texts from the pdf files and the system code from NS3451 are the features and labels that will be used for training the model, as was stated very explicitly at the beginning of the project. The document NS3451 includes numerous kinds of system code as well as their definitions. Therefore, it is natural to investigate the various types of classifications in order to find the specific type of classification that aims to target those system codes. This can be done by looking at the various types of classifications. There are a few different ways to classify things, including the binary, multi-label, and multi-class methods. In binary classification, the outcome can be either true or false, 0 or 1, black or white, and so on.

Other possible outcomes include these: Because of this, the outcome is restricted to just two labels, out of which one is selected based on the particular feature that was provided. In multi-label classification, each sample and piece of text in the is given a specific label or set of labels. One single input, for instance, may be associated with more than one label. When carrying a multi-class classification, each sample is only given a single label, or in my case, a single system code. As a direct consequence of this, a multi-class classification approach is being utilized in this project. This is as a result of the fact that each sample is only linked to a single system code originating from NS3451.

In classification, we are typically provided with data consisting of a number of attributes, of which one is utilized as a label and the others are utilized to train the model. In the final step, we put the model to the test using data sets that have never been examined before. These data sets also have labels, and we compare them to the initial label using the information from the model evaluation section. It is important to note that the testing phase of the classification model does not make use of any of the data sets that were utilized in the training phase of the model.

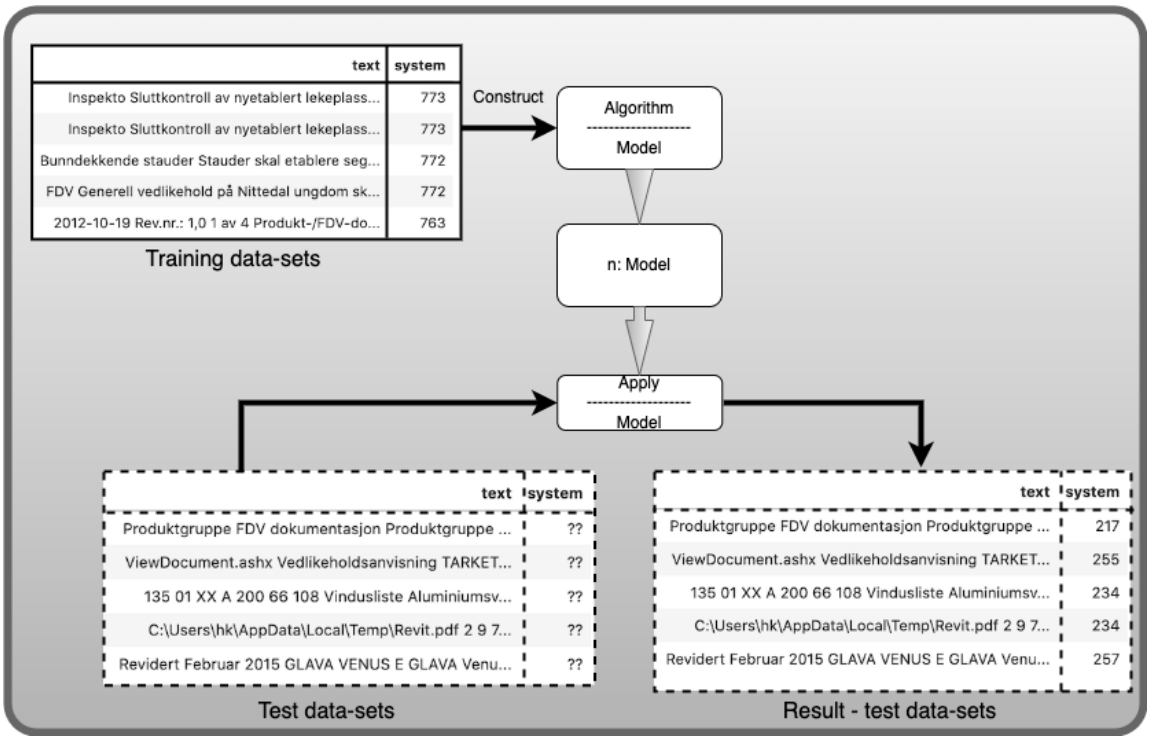


Figure 2.1: Classification structure

The classification structure shown above is used by many classification techniques.

Some examples include decision trees, Naive Bayes, Support Vector Machines, and Neural Networks.

2.3.5 Decision Tree Classifier

The Decision Tree is a well-known and user-friendly algorithm for visualizing the decision-making process, and it can be applied to the classification of more than one class of things. In addition, in contrast to many other classifications, Decision Tree is capable of handling both categorical and continuous data sets for training. In addition to this, the structure of the algorithm includes a root node, branches, and a leaf node. Each and every decision that is produced by the algorithm takes two or more different courses through the branches. The procedure starts at the root node and continues all the way out to the tip of the branches before returning to the beginning of the possibilities space.

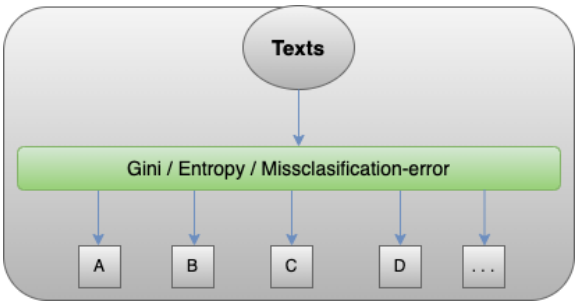


Figure 2.2: Structure of decision tree

The process of breaking up is based on gini, entropy, or missclasification error, which helps in determining a splitting point while simultaneously achieving information gain. This is illustrated in the figure that is located above. Before dividing the data set and attempting to predict a lab or system code, these methods also provide an indication of the degree to which the input data set is pristine. In order to get a better idea of how the impurity calculation is performed in general, let's us pretend that we have a list of components that correspond to a machine.

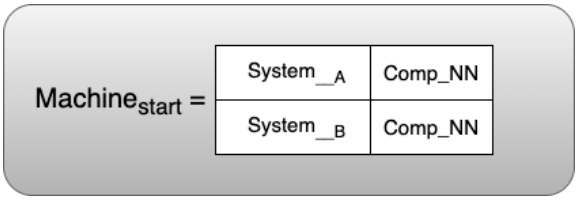


Figure 2.3: Before Splitting

if we divide the components of the machine to which they belong, we get the following:

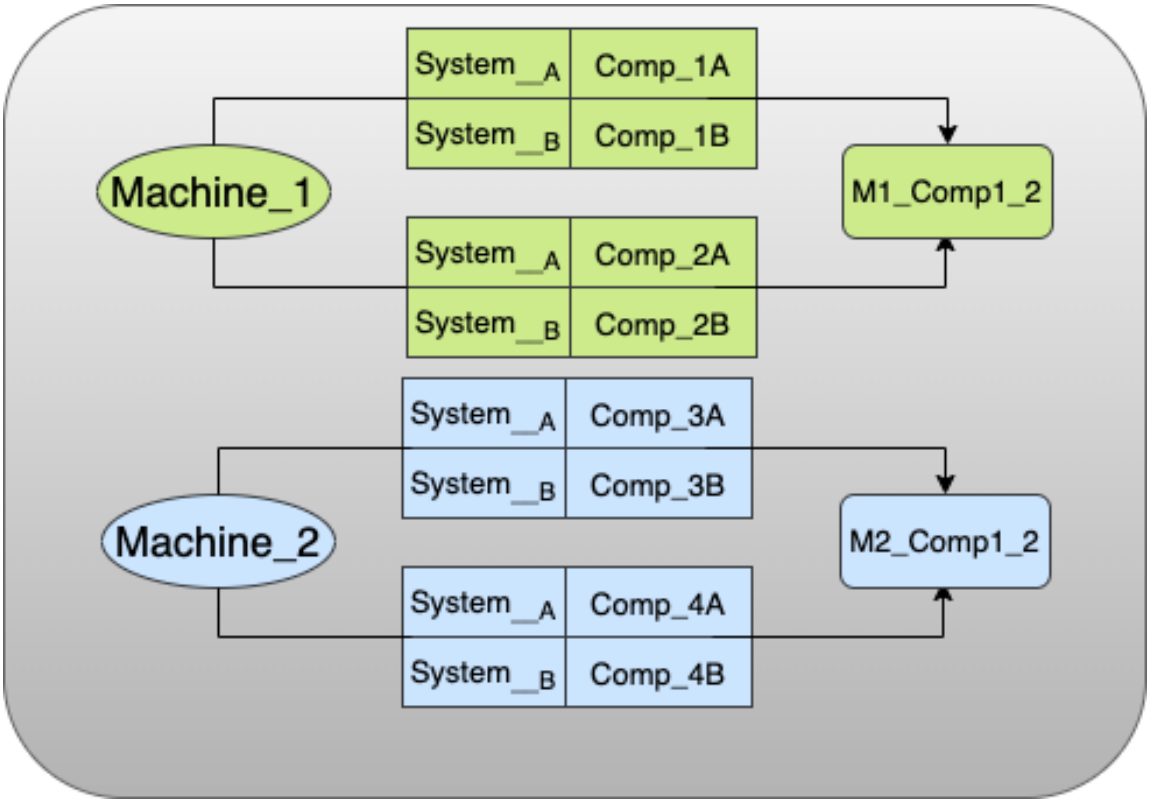


Figure 2.4: Splitting

$$GAIN_{one} = Machine_{start} - Machine_1$$
$$GAIN_{two} = Machine_{start} - Machine_2$$

The difference between before and after splitting can be seen in the image above. The gain is then archived by $max(GAIN_{one}, GAIN_{two})$. That is, we choose the attribute with

the highest gain. Let's take a closer look at the formula and some examples. And the first formula to measure impurity is GINI:

$$GINI = 1 - \sum_{n=1}^{\infty} [p(j|t)]^2 \tag{2.1}$$

where $p(j|t)$ represents the relative frequency of class j at node t [5]. If we see back to figure above in "Machine_1" and compute with given number for "Comp_1A" = 2, "Comp_1B" = 3, "Comp_2A" = 1, "Comp_2B" = 4, here, I assume that "Comp_1A" and "Comp_1B" are part of a bigger component in "Machine1_1". "Comp_2A" and "Comp_2B" are also another bigger component in machine_1. the calculation of GINI would be:

$$\Rightarrow \quad P(Comp_{1A}) = 2/5, \quad P(Comp_{1B}) = 3/5$$
$$\Rightarrow \quad GINI_{1A\_1B} = 1 - P(Comp_{1A})^2 - P(Comp_{1B})^2 = 1 - (2/5)^2 - (3/5)^2 = \underline{0.48}$$

$$\Rightarrow \quad P(Comp_{2A}) = 1/5, \quad P(Comp_{2B}) = 4/5$$
$$\Rightarrow \quad GINI_{2A\_2B} = 1 - P(Comp_{2A})^2 - P(Comp_{2B})^2 = 1 - (1/5)^2 - (4/5)^2 = \underline{0.32}$$

From the result above GINI_2A_2B is a best choose among these two. This is because, the GINI ranges from 0 to $(1 - 1/n_c)$ where n is the number of classes. If the number of classes is 4, the maximum range of GINI can have is $1 - 1/4 = 0.75$. It is also important point out that if either "Comp_1A" or "Comp_1B" has been 0, then GINI would be 0 and it tells that $GINI_{1A\_1B}$ is pure split which means it provides valuable information. And opposite, if the result is closer to the maximum range, the impurity increases. When it comes to GINI_split which is mostly used in another algorithms of decision tree such as CART, SLIQ and SPRINT, the splitting involves both the number of records in the parent and child node:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} \times GINI \tag{2.2}$$

where:
n_i is the number of records at child i and n is the number of records at node p [6]
Here the quality of splitting goes to multiplication by the ration of n_i and n. From the previous example we have parent node (GINI_1A_1B, GINI_2A_2B)and their GINI is $1 -$

[5] `https://spark.apache.org/docs/1.4.1/mllib-decision-tree.html`
[6] `https://spark.apache.org/docs/1.4.1/mllib-decision-tree.html`

$(5/10)^2 - (5/10)^2 = 0.5$ and it is also known that GINI_1A_1B and GINI_2A_2B are 0.48 and 0.32 respectively. So $GINI_{split}$ is:

$$GINI_{split} = 5/10 \times 0.48 + 5/10 \times 0.32 = \underline{\underline{0.4}}$$

Because the result is less than the GINI of the parents, it is preferable to split. In theory, if another node could be provided that is purer than this one (in a larger data set with different types of permutation), we would split based on the minimum value obtained by the GINI split.
Another type of splitting is Entropy. This operation is based on information gain.

$$ENTROPY = -\sum_{i=1}^{k} p(j|t) \times logp(j|t)$$

where $p(j|t)$ is again the relative frequency. It say - how many records belongs class j with respect to node t. Unlike GINI, the ranges of Entropy is from 0 to $log_2 \times n_c$, where n_c is the number of classes. Entropy is similar to GINI, except the fact they both have different formulas, especially the minus sign used in Entropy to avoid having negative results. If we see with example used in computing GINI:

$$\Rightarrow \quad P(Comp_{1A}) = 2/5, \quad P(Comp_{1B}) = 3/5$$
$$\Rightarrow \quad Entropy_{1A\_1B} = -(2/5)log_2 \times (2/5) - (3/5)log_2 \times (3/5) = 0.2922 \approx \underline{\underline{0.3}}$$

$$\Rightarrow \quad P(Comp_{2A}) = 1/5, \quad P(Comp_{2B}) = 4/5$$
$$\Rightarrow \quad Entropy_{2A\_2B} = -(1/5)log_2 \times (1/5) - (4/5)log_2 \times (4/5) = 0.217322 \approx \underline{\underline{0.22}}$$

even the specific results are changed, $Entropy_{1A\_1B}$ is greater than $Entropy_{2A\_2B}$ as $GINI_{1A\_1B}$ do compare to $GINI_{2A\_2B}$. Now, since the the entropy is calculated, it is possible to split based on the information gain.

$$GAIN_{split} = ENTROPY - \sum_{i=1}^{k} \frac{n_i}{n} \times ENTROPY(i) \tag{2.3}$$

This is very similar to $GINIsplit$. The goal here is to select the maximum GAIN split or the greatest reduction in Entropy. And the basic calculation is $GINIsplit$ if the formula parameters are followed. Misclassification Error, the final splitting method, measures an error produced in each node.

$$MisClassificationError = 1 - max(p|t) \tag{2.4}$$

The minimum and maximum values of the $Misclassification Error$ are same to those of the $GINI$. You can also use the formula to determine the relative frequencies by using the $p(p|t)$ notation. If it were absolutely pure, the maximum relative frequency would be 1, the error would be 0, and the minimum range would also be 0. Since it would be completely pure, the maximum relative frequency would be 1. Without getting too deep into the specifics, we can say that computing the misclassification error can be done in the same way as it was done before split methods were developed.

If, after transporting out one of the methods for splitting, we then return to the diagram labeled 2.2. The system code that the texts would be assigned to is represented by the letters A, B, C, and D. These letters are in alphabetical order. During the process of DT's operation, there are an enormous number of opportunities to improve the performance of the algorithm by using hyper-parameters; in fact, some of these opportunities are already provided by default. There are many different parameters, such as tree-depth, max-feature, max-leafe-node, and so on, that can be altered to improve the performance of the algorithm. The method of making improvements can shift from one situation to another. When a tabular data set is used, the training and improvement algorithm will be more intuitive. However, when text is used as input, the algorithm will be less intuitive. This is due to the fact that, as was explained in the introduction, in order for the algorithm to comprehend the input, the input (which consists of texts) must first be converted to vectors. It is difficult to sketch even the smallest part of the process due to the fact that the matrices can be very large and have few details. In general, the process that concludes the decision tree can take place in a number of different ways. For instance, this can be done by checking to see if all of the records belong to the same class, if so, this can be done when the values of the attributes of the given records are comparable. Alternatively, this can be done by implementing early stopping, when there is a hint that causes overfitting, and so on.

2.3.6 Random Forest Classifier

Several decision trees are used in the Random Forest classifier algorithm. The following is an example:

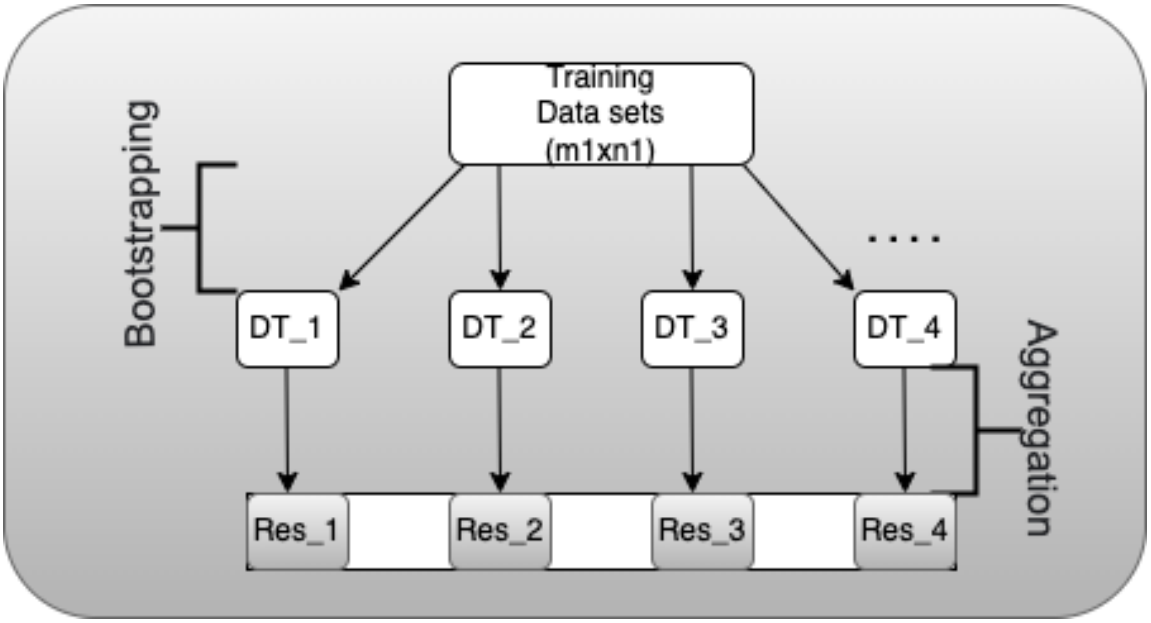


Figure 2.5: Random Forest Classifier

The process of training Random Forest involves picking records and columns at random from the sets of data used for training. To be more specific, the number of records and features that were selected for the decision trees is lower than the number of initial training data sets.

When the process of training is complete, the classification testing data set is created by selecting the majority of the outcomes obtained from bootstrapping through the decision tree models and aggregating them. This is done after the data set has been created.

2.3.7 Naive Bayes Classifier

Another classification algorithm is Naive Bayes. The classification is founded on probabilistic logic, specifically the Bayes theorem. The Bayes theorem is as follows:

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)} \tag{2.5}$$

- $p(A|B)$ is posterior probability.
- $p(B|A)$ is likelihood probability.
- $p(A)$ prior-probability.

- $p(B)$ marginal probability, which is constant for given data sets

where $p(A|B)$ and $p(B|A)$ are the conditional probabilities of events A and B. For example, the probability of an event A occurring when we already know that event B has occurred can be calculated as follows:

$$p(A|B) = \frac{p(A \cap B)}{p(B)} \quad and \quad p(B|A) = \frac{p(B \cap A)}{p(A)} \tag{2.6}$$

$$where, \quad p(A \cap B) \Leftrightarrow p(B \cap A)$$

$$p(A|B) \times p(B) = p(B|A) \times p(A)$$

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)} \qquad q.e.d$$

As states above the algorithm starts by calculating the conditional probability of events and then Bayes theorem. Let's first see how this implemented to the classification of supervised data sets and then show by simple example. In every record of the data set, there is outcome which is the result of attributes:

$$X = \big\{x_1, x_2, x_3 ... x_n\big\}, \big\{y\big\}.$$

If the data sets are now replaced in Bayes theorem given above, it would look:

$$p(y|x_1, x_2, x_3 ... x_n) = \frac{\Big(p(x_1|y)(x_2|y)(x_3|y)...(x_n|y)\Big) \times p(y)}{p(x_1, x_2, x_3 ... x_n)}$$

$$\Updownarrow$$

$$p(y|x_1, x_2, x_3 ... x_n) = \frac{\prod_{i=1}^{n} p(x_i|y)}{p(x_1, x_2, x_3 ... x_n)}.$$

During computing the probability of the records in the same data sets, the denominator shown above is the same for all. Thus, Bayes theorem can be written:

$$p(y|x_1, x_2, x_3 ... x_n) \propto \prod_{i=1}^{n} p(x_i|y)$$

And the final result for the input data set would achieved by "argmax", which means picking the maximum values of the calculated probabilities.

$$y = argmax\Big(\prod_{i=1}^{n} p(x_i|y)\Big)$$

Here is an example of NLP that helps to demonstrate the Bayes theorem mentioned above.

Description	System_code
System for cooling outdoor areas.	735
System for cooling indoor areas.	723
This is texts for systems.	735

Figure 2.6: Sample sentences

After performing some label-encoding in ”System code” and some simple prepossessing in ”Description,” the representation of the table above can be transformed to:

	area	cool	indoor	outdoor	text	system	
Sentence 1	1	1	0	1	0	1	1
Sentence 2	1	1	1	0	0	1	0
Sentence 3	0	0	0	0	1	1	1

Figure 2.7: Sample vectors

The following words appear frequently - vector forms of sentences one, two, and three. In ”Description,” ”1” indicates that the word is present in the given sentence/sentences, while ”0” indicates that the word is absent from the sentence/s. ”735” and ”723” are label encoded to 1s and 0s in ”System code.” Texts can be vectorized in a variety of ways, but

for simplicity, BOW (bag of words in binary) is used. Every sentence, once again, is made up of words.

So, using the Naive Bayes Classifier, we can determine the probability of a given sentence having "System-code" equal to 735: $where, \quad System\_code = SC$

$$p(SC = 1) = 2/3$$

$$p(SC = 1|Sentence) = p(SC)\Big(p(word\_1|SC = 1) \times p(word\_2|SC = 1) \times p(word\_3|SC = 1) \times ...p(word\_n$$

- - Count word frequency of a word in every sentence, lets say m.
- - Every time m is equal to the output(enabled system code), count n
- - Then there is n/m which represent every word

$$p(SC = 1|Sentence) \propto \prod_{i=1}^{n} p(word_i|1)$$
$$= 2/3 \times 1/2 \times 1/2 = 1/6$$

$$p(SC = 0|Sentence) \propto \prod_{i=1}^{n} p(word_i|0)$$
$$= 1/2 \times 1/3 \times 1/3 = 1/36$$

Then normalizing,

$$p(SC = 1) = \frac{1/6}{(1/6) + (1/36)} = 6/7 \quad and,$$

$$p(SC = 0) = 1 - 6/7 = 1/7$$

Probability of the given sentence to be "735" is:

$$y = argmax\Big(6/7, 1/7\Big) = 6/7 \approx \underline{\underline{0.86}}$$

The Naive Bayes algorithm is widely considered to be one of the most widely used ML algorithms for text classification. The fact that this classification makes use of an algorithm that computes the probability of each word and its output is the primary benefit of doing so. The output will be incorrect if any words essential to the word representation are omitted, which is a negative gearing.

2.3.8 Support Vector Machine

A widely used ML algorithm for supervised data sets is the support vector machine (SVM). Classification is the most common application for it, but it can also be utilized for regression analysis. A *main hyper-plain* and additional *two hyper-plains* that run parallel to the main hyper line are responsible for managing the classification of data sets in an SVM.

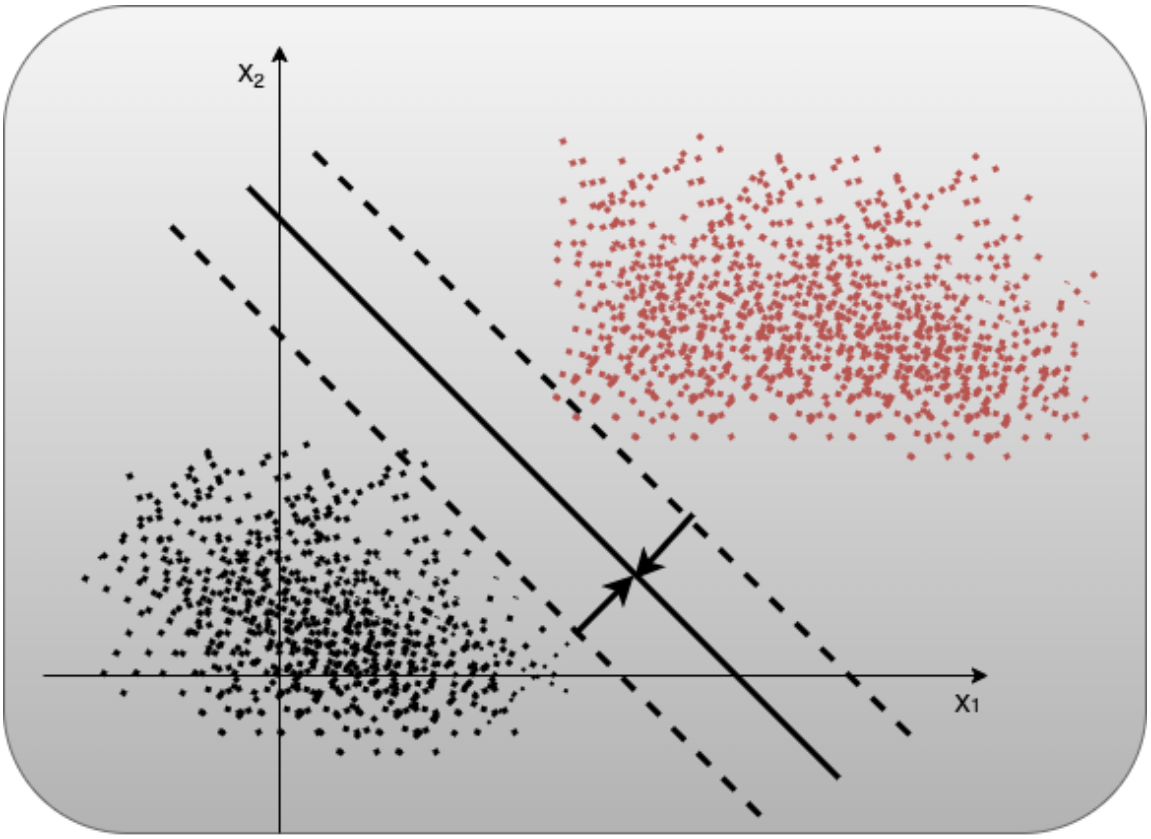


Figure 2.8: Support Vector Machine

The diagram above only applies to two classes that are almost linearly separable - data sets separated by a straight line. The *marginal distance* is the distance between the outer hyper-plains and the main hyper-plain. The algorithm with the greatest marginal distance is preferred during classification. In other words, whenever a hyper-plain is created, the one with the greatest margin should be chosen. Because the smaller the marginal distance, the greater the classification error. And the greater the marginal distance, the more generalizable the model. Errors occur when data (points in the figure) is displaced from the outer

hyperplane or classified to a different class. However, when the points are close together or pass through the hyper-lines, it is referred to as *support vectors*.

In this case, svm is calculated using a formula for a straight line that passes through one axis with slop. For example, if we want to classify folder names based on whether their system code exists or does not exist, as the project's initial problem states:

$$w^T x + b = exist \tag{2.7}$$

$$w^T x + b = not\_exist \tag{2.8}$$

where, exit and not exist can for example interpreted respectively 1 and -1, w^T is a slop of the main hyper-plane and x data and b is an intersection in x_2. If we also assume, all red points -1 and the black points belong to 1. Then by summing up and removing the w^T:

$$\frac{w^2(x_2 - x_1)}{\parallel w\parallel} = \frac{2}{\parallel w\parallel}$$

i.e $\frac{2}{\parallel w\parallel}$ need to maximize and the their optimization is:

$$y_i \times w^T x_i + b_i \geq 1$$

Since the multiplication of:

$$y_i = 1 \quad and \quad w^T x + b \geq 1$$
$$y_i = -1 \quad and \quad w^T x + b \leq 1$$

gives a positive. So at anytime when the result happen to be -1, then the misclassification error number increases. And finally the svm can be found by using the minimum of the optimization(opposite to the maximum optimization):

$$(w^* \times b^*) = min \frac{\parallel w\parallel}{2} + C_i \sum_{i=1}^{n} q_i$$

where C_i tells about how many points can be missclassified and q_i is the sum of the errors in the marginal distance.

2.3.9 Artificial Neural-Network

An Artificial Neural Network classifier is an algorithm in which each sub-part must be explained separately. In this section, I will only cover the most important aspects of this classifier.

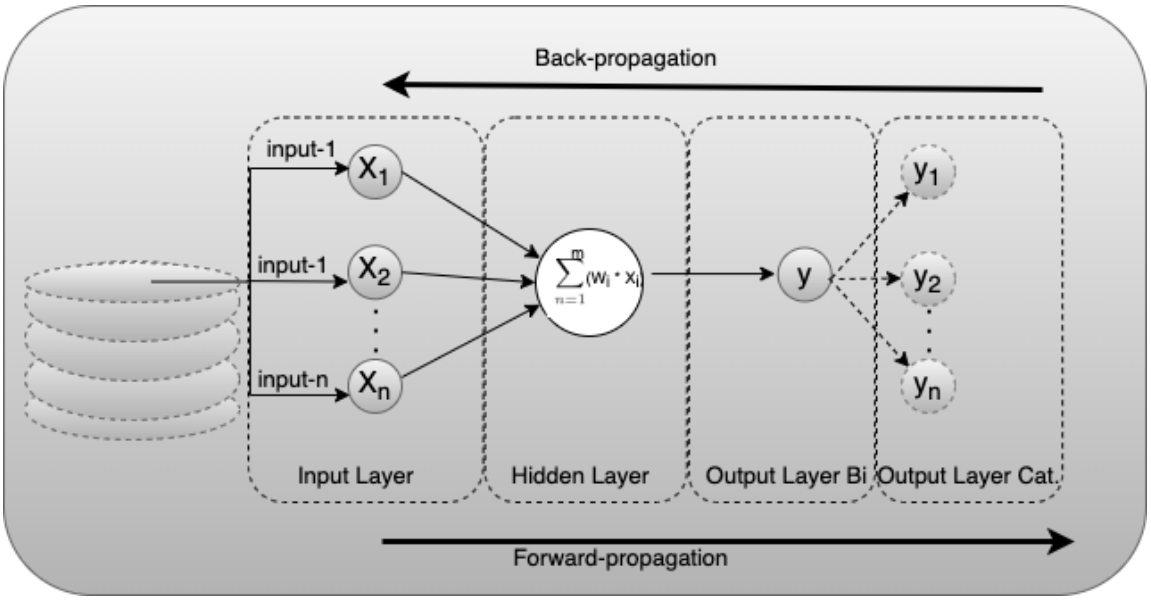


Figure 2.9: Neural Network

■ *Neurons*

The human brain's name for a neuron[7], receive input from data sets and produce output, as shown in the figure above. In this case, neuron selects a value from "input-1," "input-2," or "input-n". Hidden layer neurons sometimes provide input to the neurons. The figure above clearly shows that there are three layers (if we ignore "Output Layer Cat): input, hidden, and output. The variables in the input layer are independent and represent one record in a data set. The variables can come from either the training or testing data sets where the prediction is taking place. For validity and efficiency reasons, it is also necessary to standardize (where the mean is 0 and the variance is 1) or normalize ($minval/(maxvale - minval)$ the input variables before using them as input. [4]. The number of neurons in the outer layer can vary. It is entirely dependent on the use cases. If the expected output is binary (0 or 1, true or false) or continuous (person height), the number of neurons in the output layer can

[7] `https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414`

be one, as shown in the figure as "Output Layer Bi." If it was categorical, the neurons would be numerous - "Output Layer Cat." As a result, each iteration from the record values will only reflect its own prediction. The arrows that connect the neurons are known as weights. These weights are what determine how neuron networks learn. In other words, this is where the input values are adjusted for the next process.

■ *Activation function*

In every neuron there is a process of summation for values multiplied by its weights. And then comes the activation function.The main and most known activation functions are, Threshold , Sigmoid, Rectifier and Hyperbolic activation function.[3]

⇒ *Threshold Activation function*: In a threshold function there is an x and y function where, x-axis represent sum of multiplication for the values and their weights, as it is written in the neuron that represent hidden layer in the figure above. And y-axis number between zero and one. The function is described as:

$$f(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases} \tag{2.9}$$

This function simply explain if the value is less than zero the function will pass zero through the neuron, if the value is greater or equal to zero the function will pass one through the same neuron. And this function mainly used during the binary classification.

⇒ *Sigmoid Activation function*: The x any y-axis represents the same as the previous function. This activation function is much smother than threshold function. i.e it gives an answers as a probability, - the more closer zero it will pass zero, if it is closer one it will approximate to one. And it is mostly used in the outer layer. This function can also be use for binary classification as threshold function do. The function is written as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

⇒ *Rectifier Activation function*: Rectifier function is one of the most used function in neural networks. It's function looks as follows:

$$f(x) = \max(x, 0) \tag{2.11}$$

and it mostly used in the hidden layer, then pass to the outer layer where sigmoid or threshold is applied for final prediction.

⇒ *Hyperbolic Tangent Activation function*: This function is very similar to the Sigmoid function. Their differences lies, in this activation function the y-axis values range from minus one to 1 and it's written as

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.12}$$

- *Forward-propagation* Forward propagation is the flow where the records of the data sets goes through the steps until they arrive in the output-layer. i.e the information is entered into the input layer and propagated forward in order to get outcome values. Here it important to note that the continuation of the forwarded value and it result depend in the their input values and their weights. When the outcome values are achieved, we compare with the original value and then back-prpegate through the network. To do that, back-propagation is needed after calculating the error among the result and actual value.

- *Back-propagation* Back-propagation is an algorithm that preforms advance mathematics which helps us to adjust the weights simultaneously. Thus, we are able to know which part of the weights in neuron network belong to the given error [6]. Here, the learning rate of the algorithm will decide how the weights would be updated.

The repetition(epochs - passing through all training data sets) of these steps needed to build and train neural networks.

2.3.10 K-Means

K-means is a clustering algorithm, which mainly used unsupervised type of ML. This algorithm is applied for different scenario including text/document classification based on, for instance categories in the given clusters.
The working process for the K-mean algorithm can be defined in a few steps:

- The number of clusters or groups for the given data can be decided at the first step of the algorithm. As it's name says the "K" represents the number of clusters.

- We then select randomly "X" data points which is equal to the number of K. Here, there is an assumption that the randomly selected points are *centroids*.

- We measure the distance between each points and the selected X centroids. The distance measurement involves either euclidean or Manhattan distance, it depends what kind of dimension we are working with.

- Each points assign to the clusters, based on the calculated distance.
- The mean value is calculated whenever a new data point appears in the clusters.

All the above process, from point three to point five repeated until the given number of iteration in the algorithm is done. Another thing to point out here is that even the selection of K happen in the first step, it doesn't mean that is the best K. To find out the best K after some iteration, we can use the *elbow* method. In the elbow, we pick the K with the most decreasing in variation. This is because the quality of the clusters/groups can be checked by adding the variance of each cluster. This means when the K value increases, there is a decrements or constant value of variation.

2.3.11 Expectation Maximization - EM

EM algorithm mostly used for unlabeled data sets. EM together with MultinomialNB used in semi-supervised part of the project. The theoretical part will cover how to use for unlabeled data points. For example, if it is given many data points that belongs for two clusters for simplicity reason.

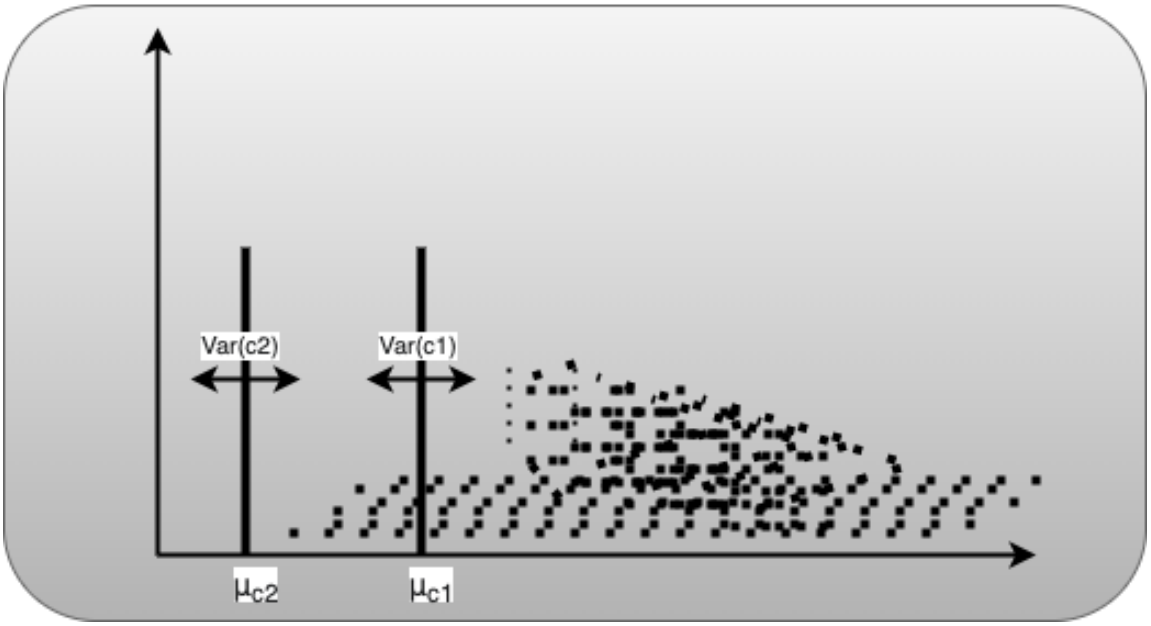


Figure 2.10: Randomly picked mean and variance

The classification of the points to those clusters starts by placing the Gaussian in a random position. i.e picking random mean and standard deviation, which in this case two mean and two variances values as a number that represents each clusters(Note: it is only one dimension example). The processes of the algorithm starts by finding out - how likeli that each points come from first cluster by the following equation:

$$p(x_i|c1) = \frac{1}{\sqrt{2 \times \pi \times \sigma_{c1}^2}} \exp^{\frac{-\left(x_i - \mu_{c1}^2\right)}{2 \times \sigma_{c1}^2}} \tag{2.13}$$

where x_i, μ, σ^2 respectively represents the data points, mean and variance for the first cluster. And this gives the probability of the point x_i come from the first cluster. Here, if the randomly picked mean and variance for these two clusters are at the most left side as it is shown in the figure, and the probability for the most right point will give small value even smaller for the cluster with mean far away to the left. There is a need to use the same equation for the second cluster, then use the Bayesian posterior that involves both equations to find out the probability that each point belongs to one of the clusters.

$$p(c1|x_i) = \frac{p(x_i|c1) \times p(c1)}{p(x_i|c1)p(c1) + p(x_i|c2)p(c2)} \tag{2.14}$$

As one can see in the formula, there is an assumption for prior probabilities of both class(brief explanation for prior in Naive Bayes section), $c1$ and $c2$. Sine the equation above gives the probability of $c1$, $c2$ can be calculated:

$$c2 = 1 - c1$$

Once finding the probability of the specific cluster is completed, which is also known as *expectation* part of the process, the *maximization* part of the given clusters preformed by the follow equations:

$$\mu_{c1} = \frac{\sum_{i=1}^{n} c1_i \times x_i}{\sum_{i=1}^{n} c1_i}$$

$$\sigma_{c1}^2 = \frac{\sum_{i=1}^{n} c1_i (x_i - \mu_{c1})^2}{\sum_{i=1}^{n} c1_i}$$

$$\mu_{c2} = \frac{\sum_{i=1}^{n} c2_i \times x_i}{\sum_{i=1}^{n} c2_i}$$

$$\sigma_{c2}^2 = \frac{\sum_{i=1}^{n} c2_i (x_i - \mu_{c2})^2}{\sum_{i=1}^{n} c2_i}$$

Here is again important to note that the probabilities of $c1_i$ and $c2_i$ lies between 0 and 1 and it helps for adjusting. The above shown example represents only one iteration which includes the estimation of mean and variance of both clusters. The estimation for the priors can also be done by adding up the probabilities $c1_i$/$c2_i$ and divide by the total. Generally the processes of EM done with several iteration until convergence.

2.3.12 Model Evaluation

This project's module evaluation is primarily concerned with the classification problem. The evaluation will provide answers to the following questions:-

- How to evaluate the performance of a model?
- What are the methods for estimating these measurements?
- How do we compare the performance of different models?

Metrics for evaluating performance frequently begin with general accuracy. When the given data sets are balanced, accuracy is used to evaluate the model. The performance is typically measured using test data. It can also be done in training data to see how well the model learns from the training data set. Because the labels in my data sets are numerous, I prefer to use two classes to demonstrate the indication of using and calculating accuracy.

	PREDICTED CLASS		
ACTUAL CLASS		Yes	No
	Yes	TP	FN
	No	FP	TN

Figure 2.11: Confusion matrix for two classes

The figure above shows a Confusion matrix of two classes for actual, which also known as ground truth and predicted part of the classes. TP, FN, FP and TN stands for True Positive, False Negative, False Positive and True Negative respectively.

- TP - tells that the actual class label is ”yes” and it is predicted as a ”yes”.
- FN(also known as type one error) - the actual class label is ”Yes” and the predicted class is ”No”. In other word the model is falsely assigning to the negative class.

- FP(also known as type two error) - the actual class label is ”No” and it is predicted ”Yes”.
- TN - the actual class label is ”No” and it is predicted ”No”.

There is only binary classification in this case, but the Confusion matrix can be used to multi-label to see how the evaluation goes. In general, TP and TN should be maximized.

The formula to calculate accuracy is shown below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.15}$$

This formula is simple to understand and tells us how many correct predictions (TP and TN) of the instances we have compared to the total number of instances in the data. It is possible that some misinformation will be spread. This can occur when we have a large number of instances of one class over the other class or classes. Assume that the instance of class A is around 10000 and the instance of class B is only 10. The developed model will almost certainly predict class A every time it predicts. And the formula's output would be misleading because the model does not detect class B correctly. Again, 99.9 percent accuracy would be incorrect. There are several solutions to this problem. Cost Matrix is one of them. Because there is equal preference for the FP,FN, and TP in the previous example, the accuracy was high. In other words, there was no priority given to the TN, for example. So, in the Cost Matrix process, the cost of misclassification is assigned, i.e. the cost of classifying the actual class as another predicted class.

COST MATRIX	PREDICTED CLASS		
		Yes	No
ACTUAL CLASS	Yes	-1	100
	No	1	0

MODEL M1	PREDICTED CLASS		
		Yes	No
ACTUAL CLASS	Yes	450	120
	No	180	750

MODEL M2	PREDICTED CLASS		
		Yes	No
ACTUAL CLASS	Yes	750	135
	No	19	596

MODEL M3	PREDICTED CLASS		
		Yes	No
ACTUAL CLASS	Yes	500	86
	No	6	908

Figure 2.12: Cost Matrix with three Models

From the figure above the accuracy and their cost are as follows:

$$Accuracy_{m1} = \underline{\underline{80\%}}$$

$$cost_{m1} = (-1 * 450) + (100 * 120) + (1 * 180) + (0 * 750) = \underline{\underline{1173}}$$
$$Accuracy_{m2} = \underline{\underline{90\%}}$$
$$cost_{m2} = (-1 * 750) + (100 * 135) + (1 * 19) + (0 * 596) = \underline{\underline{12769}}$$
$$Accuracy_{m3} = \underline{\underline{94\%}}$$
$$cost_{m3} = (-1 * 500) + (100 * 86) + (1 * 6) + (0 * 908) = \underline{\underline{8106}}$$

- m1, m2 and m2 refers to model-1, model-2 and model-3 respectively.

From figure 2.12 there are three models with their result for accuracy and calculations of the cost of matrix. The first rule to note is that the higher the cost is, the worst the model preforming. In the calculations above the result of these models in accuracy are good. Specially the accuracy for m3 is the highest and it's cost is the lowest. So, if one would choose which to model use among these three models, m3 would be ranked at the first place. When it comes to m1 and m2, it easy to see m2 has higher accuracy compare to m1, but since the cost of m2 is higher, it is preferable to choose the m1 over the m2.

It is used in the cost matrix to either reward or punish the TP in this case. It is also possible to ignore predictions that have no meaning by putting a zero in the cost matrix. It all depends on the application. In general, this demonstrates how to adjust performance when the input data sets differ in the number of instances that represent a class.
There are also Cost-Sensitive measures the performance of a model:

$$Precision = \frac{TP}{TP + FP} \tag{2.16}$$

The precision tells, of all the instance which the classifier is predicting to be true how many of them are actually correctly predicted. In other words precision is more biased toward TP and FP

$$Recall = \frac{TP}{TP + FN} \tag{2.17}$$

Recall which is also know as sensitivity, tells of all the actual true how many of them are correctly predicted. There is also opposite for sensitivity

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

And this indicates that recall is biased towards TP and FN

$$F - measure = 2 \times \frac{(Recall) \times (Precision)}{Recall + Precision} = 2 \times \frac{TP}{2 \times TP + FN + FP} \quad (2.18)$$

F-measure(F_1) is the harmonic mean of precision and recall,i.e it gives equal weight to precision and recall. F_β is another type of F measure where it gives more weight to recall which I am not going to use in this project. It shows that F-measure is biased towards all except TN.

Finally these formulas are also preferred over the accuracy due to the way they target specific role for the given problem.

Chapter 3

Problem and its Investigation Methods

3.1 Case:

Figure 3.1 shows the primary problem's visual solution.

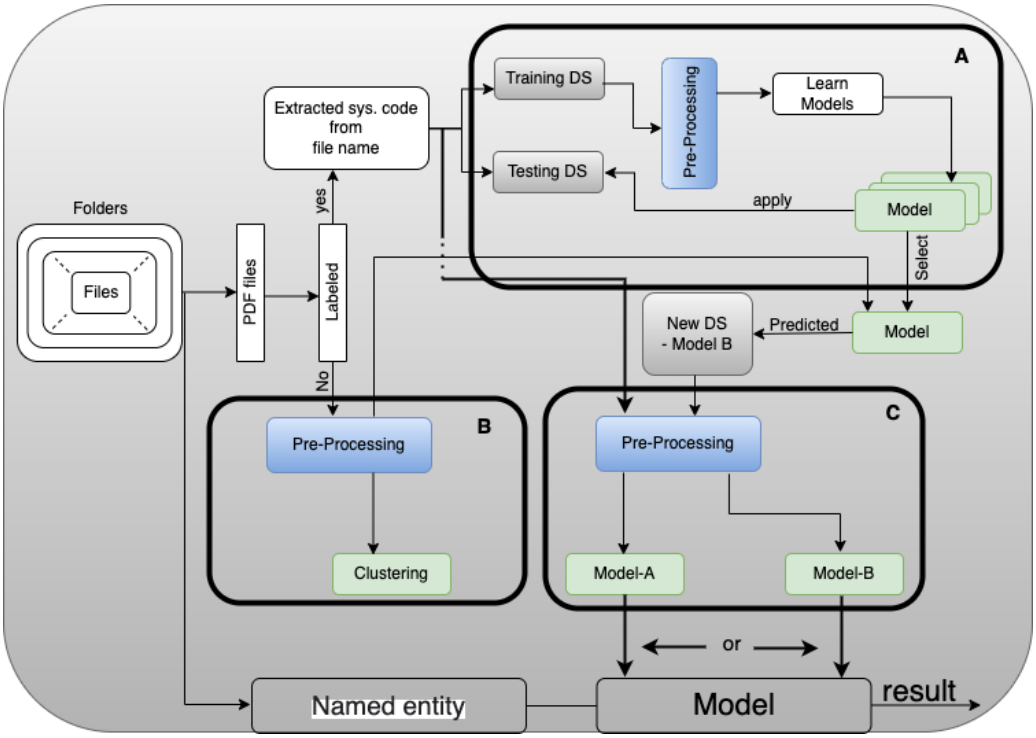


Figure 3.1: General view of the problem

The solution to the problem that is presented in this thesis can be found in the figure above. In the introduction section, stated that Autility is a company that possesses row files; however, Autility does not possess an automated system that is able to categorize files based on the system-code. Autility provided me with a number of folders that contained files in a variety of formats at the beginning of this project. The data are divided up into folders that are either structured or unstructured. Within the structured section, the label of each file can be found by looking at the title or name of the file, which is combined with the path that indicates where the file is stored within the Autility database. Therefore, in order to have a label for each file, it is essential to extract the label, which is the system-code, from the title. The system-code that is extracted from the title ought to correspond to the system-code that is denoted by the red square in figure 3.2.

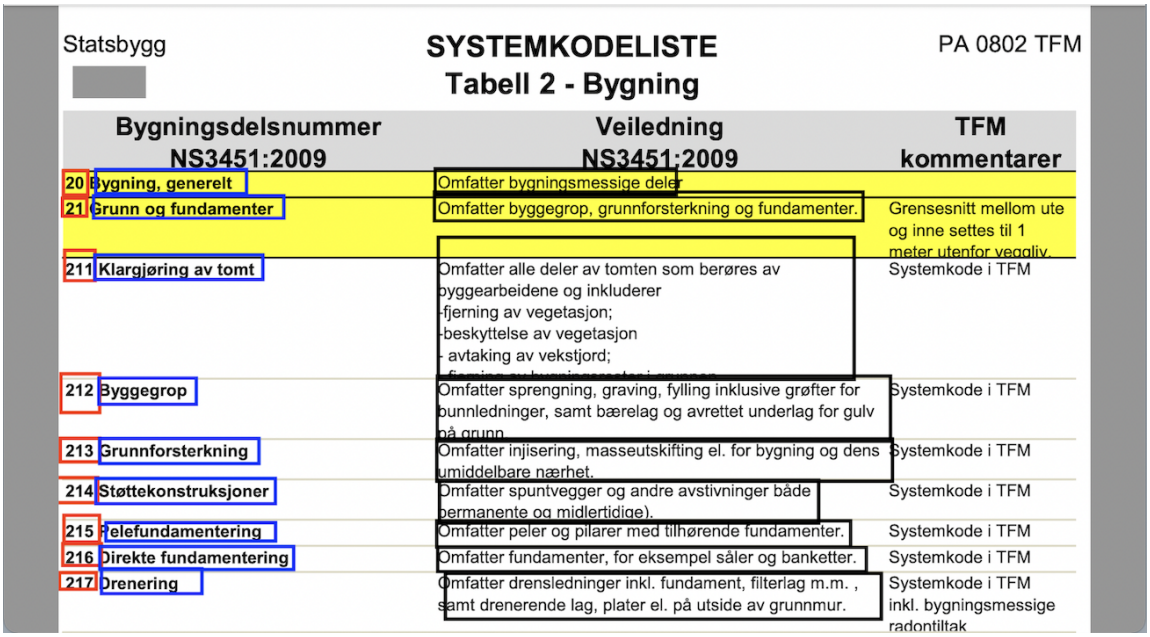


Figure 3.2: Classification structure

In other words, the first aim is to extract the label (system-code) from the title. If the label does not exist in the title, the file is saved as a file without a label,- unstructured. Here, the structured and unstructured can be interpreted as supervised and unsupervised.

In light of this, the processes of A, B, and C represent, respectively, the supervised, unsupervised, and semi-supervised types of ML. These processes are denoted by the bold letters in the figure. In part A, the dataset are separated into training and test data sets. The training part is the only one in which preprocessing and text argumentation are carried out.

Several different classification algorithms are evaluated in order to determine which one produces the best results. The preprocessing step is also included in the B part, but there is no text argumentation because the dataset is already of sufficient quality, and the primary objective of having this part is to gain some insight into the extent to which the dataset is correlated. In addition, the raw unlabeled dataset along with the raw labels are utilized in the performance of the semi-supervised Model-A in the C part. Because the original labeled dataset is so small in comparison to the unlabeled dataset, the semi-supervised form of ML is an alternative that can be looked into as an option. I also used the best model from part A to make a prediction for the system-code of the unlabeled dataset, and then I used that prediction to create a new dataset that was labeled with the system-code. Then, by combining them, Model-B (in part C) of the semi-supervised computed by giving larger percent of the data sets as unlabeled to compare the result from Model A. This was accomplished by keeping a larger portion of the dataset. Because it encompasses such a sizable and varied portion of the datasets, this project places a significant emphasis on the outcome of part C. Following the completion of these parts, particularly part C, the files are then sent on for additional processing, which involves named entities. The procedures of A, B, C, and the named entity will each receive a comprehensive breakdown in the subsequent sections.

3.2 Experiments

3.2.1 EDA - Exploring and Analysing data

There are approximately 20,000 files and the datasets was obtained through a number of different stages. Because of this, I was required to work in iterations on collections of files that were saved in a variety of formats. The following is how the number of datasets will look during the first and second phases:

```
************* FIRST PHASE STRUCTURED *************
| Total number of files are:-  1409
| DOC type -->  20 --> 1.42 %
| PDF type -->  1361 --> 96.59 %
| XLSX type -->  13 --> 0.92 %
| JPG type -->  4 --> 0.28 %
| PNG type -->  9 --> 0.64 %
| 0.14 %
************************************************
************* SECOND PHASE STRUCTURED ************
| Total number of files are:-  151
| DOC type -->  6 --> 3.97 %
| PDF type -->  130 --> 86.09 %
| XLSX type -->  1 --> 0.66 %
| JPG type -->  0 --> 0.00 %
| PNG type -->  0 --> 0.00 %
| 9.27 %
************************************************
************* FIRST PHASE UNSTRUCTURED ***********
| Total number of files are:-  8146
| DOC type -->  281 --> 3.45 %
| PDF type -->  5840 --> 71.69 %
| XLSX type -->  46 --> 0.56 %
| JPG type -->  225 --> 2.76 %
| PNG type -->  52 --> 0.64 %
| 20.89 %
************************************************
************* SECOND PHASE UNSTRUCTURED **********
| Total number of files are:-  8206
| DOC type -->  555 --> 6.76 %
| PDF type -->  5607 --> 68.33 %
| XLSX type -->  49 --> 0.60 %
| JPG type -->  510 --> 6.21 %
| PNG type -->  439 --> 5.35 %
| 12.75 %
************************************************
```

Figure 3.3: The number and types of files in first and second phase

Note: There are labeled files in the unstructured part of the data sets as well.

The original data sets were mixed, both in terms of file type and how they were assigned to system codes. The terms ”structured” and ”unstructured” refer to whether the data sets were labeled or not; labeled files were present in the unstructured portion of the datasets. I used the Python library Pandas in Jupyter Lab to explore those files. To do so, I needed to first organize the files and create data-frames in pandas for subsequent processes. Pandas allows for analysis in addition to creating data-frames, such as seeing the number of rows and columns, data types, the number of unique values in different columns, and so on. Another significant advantage of the library is the ability to run a single or small snippet of code rather than a whole method/function or class written in Python.

According to what is shown in the figure that is located above, in the first phase, I have 9555 that are both manually structured (1404), as well as unstructured (8146). By manually structuring the data, each file was assigned a system code from the “SYSTEMKODELISTE NS3451” database, whereas the unstructured data set was simply organized into folders. During the second phase, the data sets contained roughly the same number of files as the first phase, but unstructured data sets were the predominant type. During both rounds, the most common types of files were pdf, doc, xlsx, and jpg, with a small percentage containing other file types. The work is primarily with readable pdf files because the percentage of PDF files was significantly higher than that of the other types of files. The next step is to combine the data sets into the respective categories. Figure 3.4 shows the arragment of the data sets.

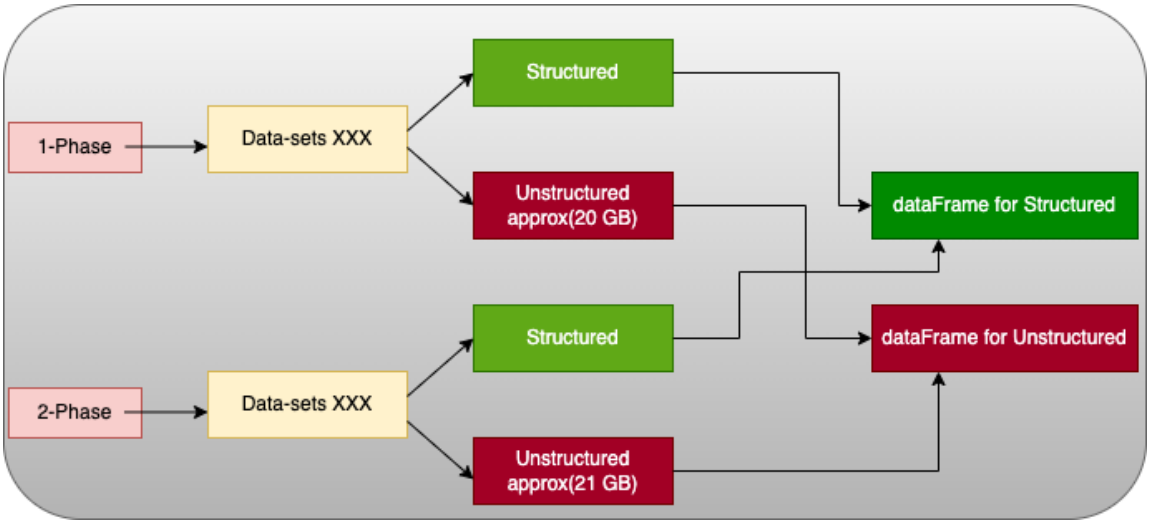


Figure 3.4: Arrangement of data sets

When all of the labeled PDF files from the structured and unstructured data sets were

merged into a single data frame, the new total was slightly higher than 2040. Following the removal of the files that did not have a valid label (system code) and languages, the number was reduced to 1417, with approximately 101 distinct labels. The first five rows are laid out as follows:

	text	system	language
0	Inspekto Sluttkontroll av nyetablert lekeplass...	773	Norwegian
1	Inspekto Sluttkontroll av nyetablert lekeplass...	773	Norwegian
2	Bunndekkende stauder Stauder skal etablere seg...	772	Norwegian
3	FDV Generell vedlikehold på Nittedal ungdom sk...	772	Norwegian
4	2012-10-19 Rev.nr.: 1,0 1 av 4 Produkt-/FDV-do...	763	Norwegian

Figure 3.5: Example from structured data set

The datasets are wildly unbalanced while also containing a staggering number of labels that are all distinct from one another. To see this, one only has to look at the diagram below. There are many more texts in system code 360 than in system code 519, for example.

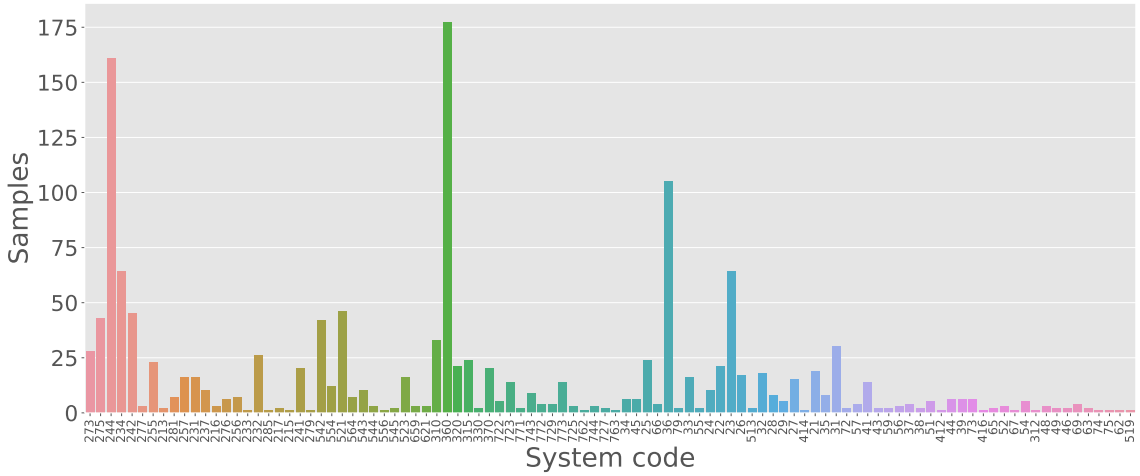


Figure 3.6: Number of text files with their labels(system code))

The variation in the number of texts in relation to the unique system code will have a negative impact on the training of the ML model. In other words, the model will not learn

enough from the input provided. To solve this problem, I tried two approaches:

- 1. Either use a large number of texts/samples to represent the system code, or
- 2. carry out *text-augmentation.* to increase the number words in each sample

These two options can help train the model as well as possible. Text augmentation is a popular method of increasing data to train the model while avoiding overfitting.

Examining the text's content is the first step recommended for resolving the issue. This step may also be of assistance in establishing which model of ML can be applied for classification. Checking the content of a text is important because even if a large number of samples have the same label, those samples might only contain a few words each. In addition to this, there is a possibility of uncertainty or an unlearned model occurring if the samples do not contain enough words.

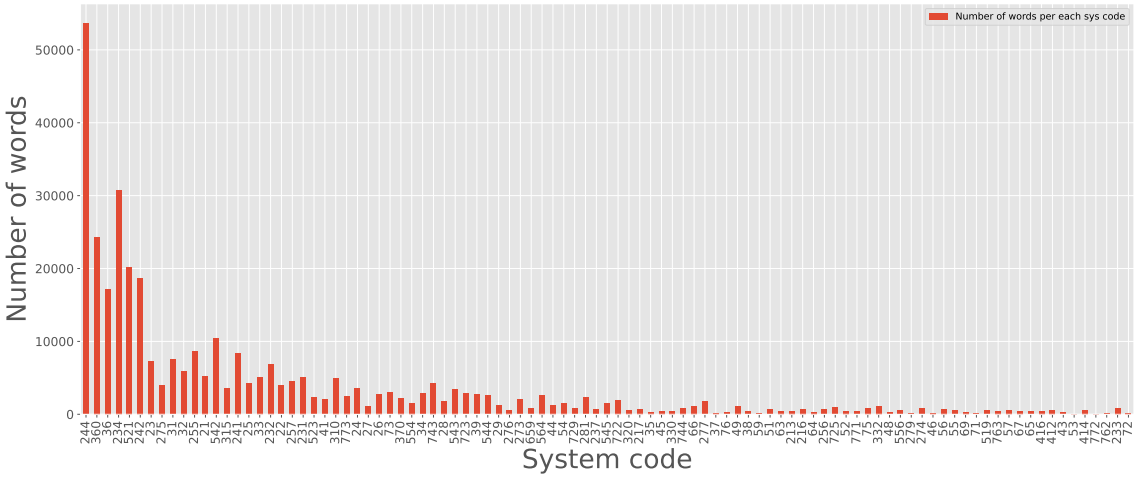


Figure 3.7: Sum of words per each system-code

From the figure 3.6 and figure 3.7 that the previously mentioned argument happen. In the event that we get a better look at system codes 273 and 244, The total number of words for system code 273 is significantly lower than the total number of words for system code 244, despite the fact that the number of samples for system code 273 is significantly higher. Following that, it's reasonable for me to wonder if the words I'm seeking for are, in fact, valuable ones.

3.2.2 Data Preparing

After cleaning the texts, there is a significant reduction in the range to the total number of words. The straightforward strategy for computing the total number of words after cleaning the texts is as follows:

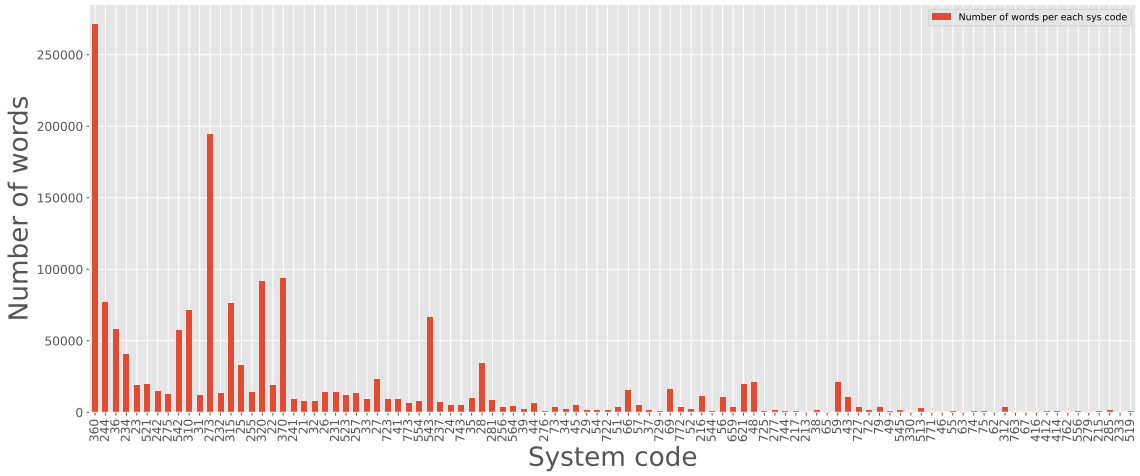


Figure 3.8: : Sum of words per each system-code after cleaning

However, due to the unbalanced nature of the data, the decision is made to increase the number of words and samples. This is true for system-code samples of five or greater. To complete the process, nlpaug's word augmentation is used. This works by randomly selecting a word from each random sample that corresponds to the given system-code, then randomly selecting a word from that sample, augmenting the word, and replacing it on the same index (same place). When there are a lot of words, they are chosen in a variety of ways. Finally, the sample is identified as a new sample, but with the same label or system code as the original sample. It should be noted that the Norwegian and English are included as data size increase reasons. The entire code can be found in the attached code-file, and a small portion of the code to do the job looks like follosws:

```
for w in random_words:
    idx = text_.index(w)
    index_agumented_word.append(idx)
for i, y in zip(index_agumented_word, agumented_words):
    text_[i] = y
new_text = " ".join(text_)
data = data.append({'text': new_text, 'system': system_code, '
   language': language_}, ignore_index=True)
```

Listing 3.1: Word Augmentation

and the end result after some rearranging looks like this:

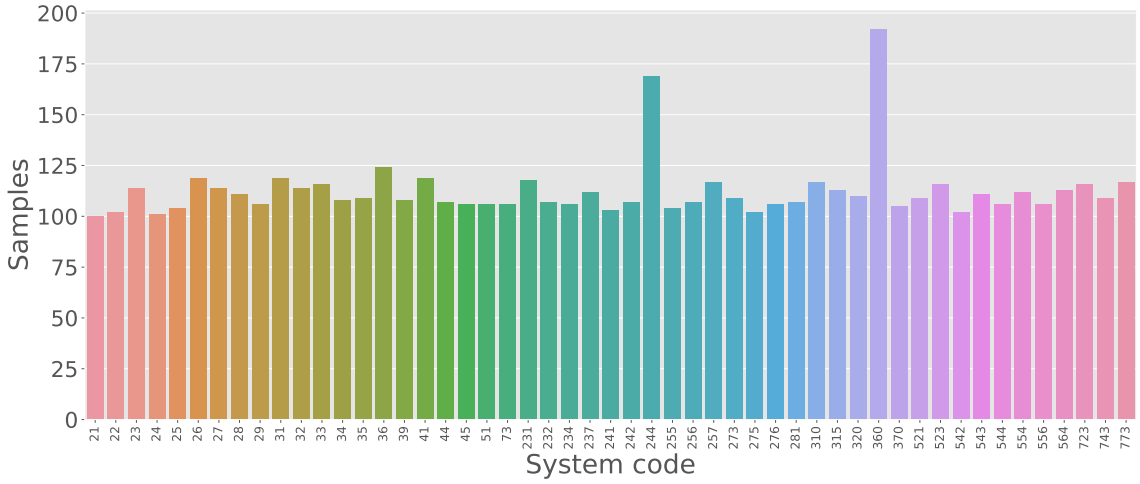


Figure 3.9: For samples larger than five, use word agumentation

As mentioned in the theory of Natural Language Processing section, there are techniques that can classify into two categories: syntax and semantics. Again, sentences are made up of words, and it is these individual words, their parts of speech, and their placement in the sentence that provide the computer with knowledge and context as to what the sentence is really trying to say.

This raises the question of how the computer knows what the various parts of speech are, as well as the question of how it can figure out all of this even if it does know the various parts of speech. The power of today's large amounts of data could provide the solution. In practice, any programmer is capable of gathering a variety of words, phrases, sentences, grammatical rules, and word structures and then feeding them into an algorithm. An algorithm can use this information to learn what words typically end up next to each other, how a sentence should be formed, why certain words fit into a sentence better than others, and so on. It can also learn why certain words fit into a sentence better than others. And this is exactly how a computer will eventually determine which word in a given sentence makes more sense than the other words. Consequently, when cleaning the texts, both the syntax and the semantics of the text are taken into consideration. This ensures that the words have meaning for the final product.

So cleaning texts involves some standard steps when working with Natural Language Techniques, such as Syntax, which is the set of rules that governs the STRUCTURE of sentences:

- A. *Tokenization*, remove *punctuation's* and change the words to lower case.
- B. Removing *stop words*.
- C. Stemming
- D. Preforming *lemmatization*.

There are two types of tokenization: sentence tokenization and word tokenization. Sentence tokenization divides a paragraph into distinct sentences, whereas word tokenization divides a sentence into distinct words. As a result, the computer is able to learn the potential meaning and purpose of each unique word.

Stemming is the process of breaking down a word into its component parts, known as its stems. This is achieved by stripping the word of universal prefixes and suffixes such as "es," "s," "ing," and "ed".Although stemming is an effective method, this straightforward chopping based solely on common prefixes and suffixes can sometimes remove necessary components of a root and result in a change in the original word's meaning. This brings us to the second argument in favor of lemmatization rather than stemming.

Instead of chopping off the beginnings and endings, lemmatizations reduce a word to its root form by morphologically analyzing the word. In other words, if the words "am", "are", and "is" are presented, the root form for these is "be" as demonstrated by lemmatization. Stemming, on the other hand, would not have figured it out because chopping off any letter of these words would not have produced "be" [1]. Thus, lemmetization on the texts is preferred, and the data frame has three features after some adjustments.

They are *"text"*, *"system"* and *"language"*.The cleaning of the text feature begins by removing all punctuation found in the *string module*

```
0 import string
1 print(string.punctuation)
2 # output:!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Listing 3.2: Code to show possible Punctuation

[1] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Taking out the punctuation during the training process is beneficial to the model. This is of the utmost importance in my situation because the low number of words contained in each file will cause the model to fail if any punctuation is included in the presentation. After that, I continue to split the text into words in each of the files so that I can identify the stop_words and perform some lemmatization. To do the job I used *tokenizer* from *spaCy*. The removal of stop_words is accomplished by determining whether a word is on the list *stop_words*.

```
from nltk.corpus import stopwords as sw
stop_words = sw.words('norwegian')
print(stop_words[:10])
# output:
                          ['og', 'i', 'jeg', 'det', 'at', 'en',
    'et', 'den', 'til', 'er']
```

Listing 3.3: Code for Using Norwegian stop words

As shown in the figure, I used the NLTK (Natural Language Toolkit) package in Python, which contains a list of stop words for several languages. I've also printed out ten of the stop words. Other words can also be used as stop words. The reasons for removing these words are to increase the learning rate of a classification model and to have a shorter runtime while training the model. This is due to the space saved by removing the stop words. Then, as shown below, lemmatization was used to meaningfully shorten the words back to their root form.

```
import spacy
nlp = spacy.load("nb_core_news_lg")

for i in nlp("Dette sier noe om vaart samfunn produserer
   enorme informasjon."):
    if str(i) not in stop_words and str(i) not in string.
   punctuation:
        print(i.lemma_)
# output: dette, si, vaart, samfunn, produsere, enorm,
   informasjon
```

Listing 3.4: Code for demonstrating Lemmeatization

When working with a data frame, the following code snippet performs all of the preceding preprocessing steps.

```
def punctuationsRemoval(self, data):
    for punct in string.punctuation:
```

```
        data = data.replace(punct, "")
    return data

def token(self, data):
    # tokenization, stop words, lemma_
    data['stop_Lemma'] = data.Text.apply(
        lambda text: " ".join(each_token.lemma_
                              for each_token in nlp(text)
                              if not each_token.is_stop))
    # remove the punctuation
    data['clean'] = data['stop_Lemma'].apply(self.
   punctuationsRemoval)
    return data
```

Listing 3.5: Code for preforming Preprocessing in Data Frame

Yes, working with natural language is difficult, and the proportion of data sets that have been labeled is small. Nonetheless, high-quality data is required, despite the fact that the size of the data set limits the ability to develop advanced models. The second option for avoiding the problem of having few and unbalanced datasets is to increase the number of samples, - as shown above with textual augmentation from the *nlpaug* python library. The texts are written in several languages. To increase the sample count, I used the Google API to translate all detectable languages to English, followed by text augmenation. However, in order to increase data size, I used both English and Norwegian at first.

```
import googletrans
from googletrans import Translator
import nlpaug.augmenter.sentence as nas

translator = Translator()
text_example = "Dette er en pr ve"

translate_eng = translator.translate(text_example, dest="en")
print("Original text: ", translate_eng.text)
aug = nas.ContextualWordEmbsForSentenceAug(model_path='xlnet-
    base-cased')
augmented_example = aug.augment(translate_eng.text, n=3)
augmented_example
# output: Original text:  This is a test ['This is a test that
     we should ask every kid at your college in an absolute
    100% positive manner.','This is a test test on you!', 'This
     is a test of patience for the last couple']
```

Listing 3.6: Code for preforming Text agumentation

The *n* refers to how many agumneted string text we can have. Because n equals 3, the result shows three arguments in the list. Then it was translated back to Norwegian. NLPaug can be Character, Word, or Sentence augmentation. These types of argumentation have several methods that can be used in doing valuable operations to increase the texts that help learning ML model. Some of the methods include synonym replacement, shuffling, random deletion, and random insertion. Because of the time constraints, word-augmentation with contextual word embedding was chosen for this project. Preprocessing was only used for training data sets when working with the original labeled dataset. This was due to the time required to translate from one language to another. It took approximately 7.5 hours to only 1/10 of the data sets, particularly with text augmentaion (90 of the approx 1000 data sets). So the result was time-consuming and tested in GPU from our university, but since using GPU in our university is primarily for those working matrices operations, I decided to work with EM, which also aimed to both labeled and unlabeled data sets. When it comes to undetectable files, they are either matrices or websites.

3.2.3 Text Vectorization

Texts must be transformed into numerical representation before they can be understood by ML models. As I discussed in the theory section, there are several methods that can be used. In this project, I used the *Tfidf-Vectorizer* from *sklearn*. Tfidf is an abbreviation for *"term frequency - inverse document frequency."*[7]. The "Tf-idf" algorithm derives its functionality from its name. The number of times a word is used throughout a document is referred to as its term frequency (tf). There are a few different ways to calculate this frequency, with a straightforward count of the number of times a word appears in a document being the quickest and easiest option. After that, the frequency can be changed according to the total length of the document or the raw frequency of the word that appears the most frequently in the document. While the inverse document frequency(idf) of the word is calculated inversely across a set of documents. This indicates how common or uncommon a word is throughout the entire set of documents. The number's proximity to zero indicates the frequency with which a word is used. To determine this metric, take the total number of documents, divide that number by the number of documents that contain a word, and then calculate the logarithm of the result. Tf-idf can then be expressed mathematically as:

$$tfidf = f(t, d) \times f(t, D)$$

anf "tf" represented by f(t,d) and "idf" by f(t,D). And t is the term and d is a single document and D is the entire document. This yields a number indicating how frequently a word

appears in a text in comparison to how frequently the same word appears in all given texts.

Pipline [2] from Scikit-Learn is used to make the code more clear. This class accepts a variety of parameters, including Tf-idf.

```
0
1 from sklearn.pipeline import Pipeline
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 kwargs = {
4     'ngram_range': (1,2),
5     'strip_accents': 'unicode',
6     'decode_error': 'replace',
7     'min_df':2,
8     'max_features':20000
9 }
10
11 # calling tf-idf, and other parameter - classification
     algorithm and kwargs
12 text_clf_nb = Pipeline([
13     ('tfidf', TfidfVectorizer(stop_words= nlp.Defaults.
     stop_words, **kwargs)),
14     ('clf_nb', MultinomialNB()),])
15 model1 = text_clf_nb.fit(X_train, y_train) # NB for example
```

Listing 3.7: Code for importing and initialization Pipeline, using Tf-idf as parameter

As the listing above shows, additional parameters can be included. Additionally, the *fit* method constructs vocabulary from given text, and another *transform* method is embedded within Pipline. If the choice of using Pipeline is ignored, it is also possible to use fit and transform as one method. The transform method returns vectors for each text. Each vector has a position for each presented word in the text, and each position has a Tf-idf frequency. The length of the created vector is equal to the vocabulary in the texts, which represents all unique words in the entire data set. The unique words do not include words such as stop words or words that are generally ignored by the algorithm. It is also worth noting that if a word appears frequently in all documents/texts or only in a few documents/texts, the approach of "Tf-idf" will be close to 0 and 1. And this will not gain in information acquisition when learning the model.

Tf-idf adds a new operation: working with multiple sequential words instead of a single word. *ngram_range* (listing above) provides the possibility of seeing the occurrence of multiple sequential words. The occurrence of multiple sequential words may have different

[2] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

meanings because the occurrence is seen as a unique word when compared to, say, a single word. As a result, the learning capability of a random classification model may improve. The obvious disadvantage of larger ngram range is that longer vectors are created, and the space occupied by these vectors requires a longer run-time. *max_features*, as shown in the figure above, can also be used to limit the length of the vectors. In cases where the vectors number in the thousands, a class called *SelectKBest* from feature-extraction modul can be used. SelectKBest enables us to have a vocabulary of our choosing and provide the words that are best represented for the given system codes. This can be accomplished by employing the previously mentioned fit method, in which the vectors and associated system codes collaborate in the model. This class is put to the test in one of my neuron network classifiers, as follows:

```
from sklearn.feature_selection import SelectKBest, f_classif
# Select top 'k' of the vectorized features.
TOP_K = 20000
select = SelectKBest(f_classif, k=TOP_K)
# Learn vocabulary from training texts and vectorize training
    texts.
X = vectorizer.fit_transform(texts)
select.fit(X, y)

# converting to numpy
X = selector.transform(X).astype('float32').toarray()
```

Listing 3.8: Code for importing and initialization SelectKBest

And this can reduce the run time during model training while having little impact on model accuracy.

3.2.4 Implementation of the Models

The data set had to be divided into training, testing, and validation before training the model. In this project only training and testing data sets is used. The goal of dividing the data sets is to see how the model performs in unseen data. The test size is 20%, with the remaining 80% used to train the model. It can be accomplished by the following snippet:

```
from sklearn.model_selection import train_test_split
X = df_cop["clean_text"]
y = df_cop['class'].astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

```
X_train.shape,X_test.shape
```

Listing 3.9: Code for splitting the given data sets into training and testing sets

The scikit-learn *train_test_split* method was used to randomly split the data sets in the given percentage, with the data sets being shuffled before the splitting process began. This process is primarily used for the supervised part(see 3.1 A-part)" of ML.

The implementation of models that use the structured portion of the dataset in conjunction with the augmented text begins with classical classifiers. It is important to note here that the supervised dataset is reduced to 3333 samples after cleaning. And, at first, the algorithms were taken directly from sklearn (with default parameters) and the order of implementation is the same as the theory section. For example, decision tree from sklearn [3]:

```
# Decision tree
from sklearn.metrics import accuracy_score
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
score = clf.score(X_test, y_test)
#preds = clf.predict_proba(X_test)
#print('Accuracy: {:.5f}'.format(accuracy_score(y_test,preds.
    argmax(axis=1))))
print('Model accuracy score with default hyperparameters:
    {0:0.4f}'. format(score))
# Output: Model accuracy score with default hyperparameters:
    0.9100
```

Listing 3.10: Code for Decision Tree with default parameters

and random forest [4], Naive Bayes [5], support vector Machine [6] and additonal KNeighborsClassifier [7] are also computed.

The model evaluation metrics for each of these algorithms are also computed in order to find the best model that can be used later, for the semi-supervised part. Performance metrics include time spent training and testing the model, accuracy, precision score, mean

[3] https://scikit-learn.org/stable/modules/tree.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[5] https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes

[6] https://scikit-learn.org/stable/modules/naive_bayes.html

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

squared error (MSE), and f1-score [8]. All of this is included in the main attached code file, and the results are displayed in the coming subsection.

In addition to the above mentioned algorithms simple neuron networks(NN) added to compare the results with the following structure:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 64)                3628160
_________________________________________________________________
dense_4 (Dense)              (None, 74)                4810
_________________________________________________________________
dropout_1 (Dropout)          (None, 74)                0
_________________________________________________________________
dense_5 (Dense)              (None, 50)                3750
=================================================================
Total params: 3,636,720
Trainable params: 3,636,720
Non-trainable params: 0
```

Figure 3.10: Neural Network Structure

The parameters in the structure can be calculated as $output\_size \times (input\_size + 1) = number\_parameters$. For example in the fist(*dense_3*), the parameters is equal to $64 \times (56689 + 1) = 3628160$ where 1 is bias. *Dense_4* and *dense_5* can also be calculated in the same way. Some other important parts which are important to tune the classifier:

- *Learning_rate* - is a step size how the process goes forward under improvement, i.e while the loss is decreasing.
- *epochs* - An iteration through the training data. And it has a major roll in learning the pattern.
- *Batch size* - decision on how many set of training group/batch should preformed in one iteration.
- *Loss* - is the difference between the actual result and the estimated result.

Data also generated other new data sets that can be used in the sem-supervised operation of this project using this ANN. This part can be explained in details later. The ANN

[8] https://scikit-learn.org/0.22/auto_examples/text/plot_document_classification_20newsgroups.html

algorithm has an input, hidden and output layers just like as it is introduced in the theory part. The number of neurons in the input layer is 56689, which is equivalent to the shape for training data sets initialized during *train_test_split* method implementation. The number of neurons in the hidden layer decided by the CV(cross validation), which gives the best result after testing the model in many different way so that the metrics measure of the model become as good as possible. If there were much larger labeled data set, the efficiency of processing the data would also be important. Thus the reason behind using Rectifier (*"relu"*) in the in both input and hidden layer also give a meaning. In the output layer, there are 50 neurons since the expected output from top 50 represented system codes. While activation function in the output layer is *'softmax'*, since softmax is preferred mostly when the expected output belong to multi-class.

Clustering the system-code is also included to see the relationship between the texts. The algorithm collects data with similar texts that are as close as possible in a given dimension. Specifically, when the algorithm computes data, it generates vectors and then sketches them in the given dimension. This can be used to determine how similar the data points are. The greater the differences in the content of the texts, the greater the distance between them. In this case, it is chosen to display the data in two dimensions.

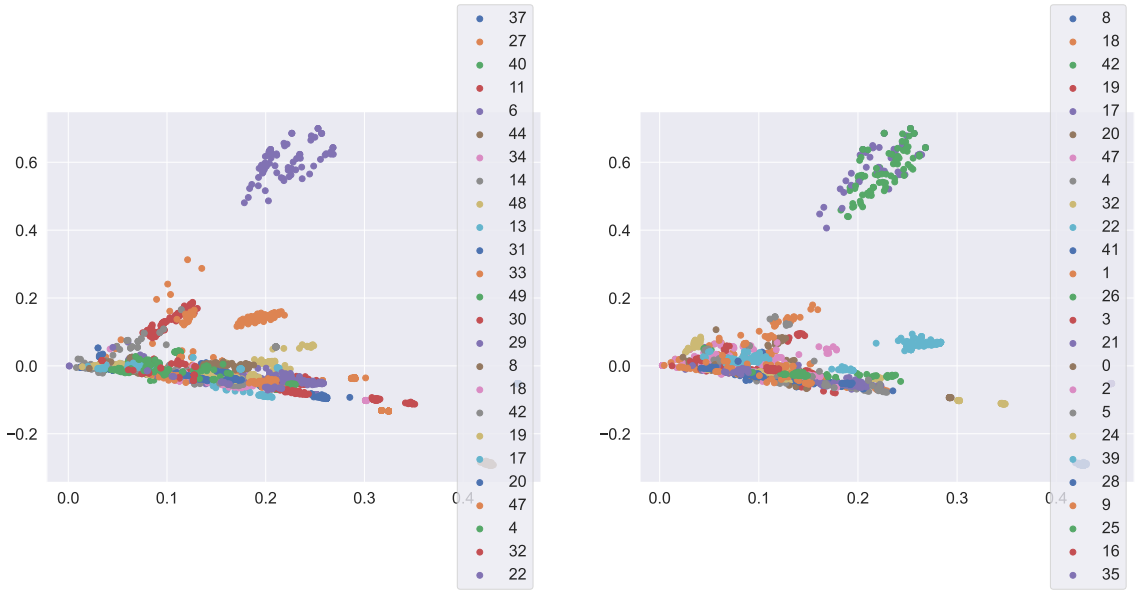


Figure 3.11: Cluster of the structured data sets

In the figure above we can see the data points overlap. This indicate that the more complicated model is needed to group the points so that they can have a clear system code

that belong to each group. Some of the data points lies far away from the others this can be due to the lack of enough words in the content of the texts or they are very different from each other. The figure represent for the top 20 at once, I could also have only few samples that represents for system code to see much clearer the similarity among the data sets. The *Truncated SVD*(above) of the first part of the top 20 and *KMeans* of all the top 20 can be represented as follows. Both Truncated SVD and Kmeans are from scikit-learn can be implemented as follows:

```
0
1 sysCode = list(df_all_structured_pdf["system"].value_counts().
    keys())
2 # converting the labels to numpy before using sklearn
3 y = np.array(labels)
4
5 # Top n most common system code or label:
6 n = 20
7 if len(sysCode) > n:
8     top_cats = sysCode[:n]
9     df_all_structured_pdf =
10         df_all_structured_pdf[df_all_structured_pdf["system"].
    isin(top_cats)]
11     sysCode = list(df_all_structured_pdf["system"].
    value_counts().keys())
12
13 # Create dictionaries for converting between category and
    numbers:
14 sys_to_num = {}
15 num_to_sys = {}
16
17 # Create lists for the text and system codes:
18 texts = []
19 labels = []
20
21 for i, cat in enumerate(sysCode):
22     sys_to_num[cat] = i
23     num_to_sys[i] = cat
24
25 for i, row in df_all_structured_pdf.iterrows():
26     texts.append(row["text"])
27     labels.append(sys_to_num[row["system"]])
28
```

```
# importing from sickit-learn
from sklearn.decomposition import TruncatedSVD

# initialising truncated model
svd = TruncatedSVD(random_state=42)

# calling fit_transform method
truncated_x = svd.fit_transform(X)


from sklearn.cluster import KMeans
from sklearn.utils import class_weight

# Compute class weight. This is used to make the model
# more likely to guess that a sample belongs to an
   underrepresented label.
class_w =
    class_weight.compute_class_weight('balanced', np.unique(
   labels), labels)

# initialising Kmeans model,, where the number of cluster is
   equal
# to the number number of top20
km = KMeans(n_clusters=len(sysCode), random_state=42)

# calling fit_transform method
K_mean = km.fit_transform(X, class_w)
```

Listing 3.11: Code for preforming TruncatedSVD and KMeans

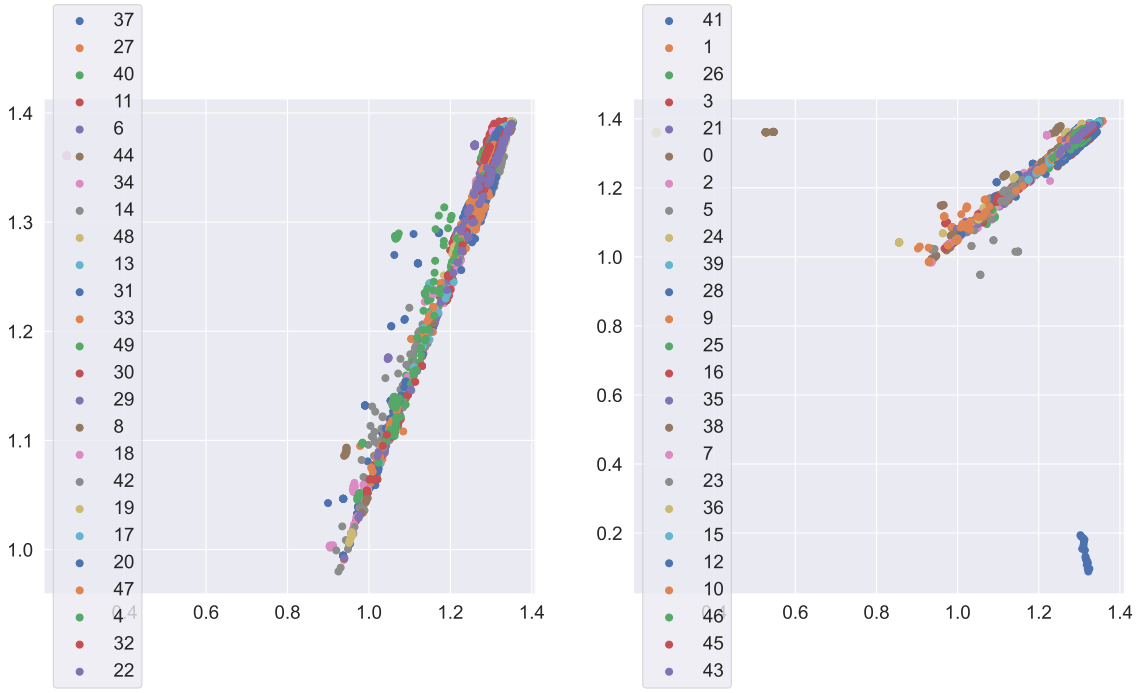


Figure 3.12: Cluster of the structured data sets

Even after running the KMeans algorithm, the data points continue to overlap with one another. This indicates that the model will have a difficult time classifying the data sets in relation to the system-code. On the other hand, as we can see in the figure that represents KMeans, the data points are arranged in an ascending line. Additionally, this will provide us with some results in terms of accuracy. Clustering in the labeled portion of the given data, in general, is an indication that the model can only learn some due to the fact that the relations between the points shown.

So far, the implementation in both labeled and general cleaning that applies for the unlabeled dataset discussed. These were a precursor to the main component of implementation in this project, semi-supervised learning. As stated at the start of the project, because the amount of labeled data received from Autility were very small, semi-supervised learning - a mixture of labeled and unlabeled data is preferable. If we return to 3.1, we can see that the process in the C section of the figure is carried out in two ways.

Let us now look at figure 3.1 C, which shows how model-A and model-B are implemented in the same part. Model-A is implemented using both Naive Bayes and Expectation-Maximization algorithms. The Naive Bayes classifier is primarily used in supervised ML, where a labeled data set is assumed. The Expectation Maximization algorithm is used for

unsupervised ML when the data set fed to the algorithm is unlabeled. In the theory section, the working process of these algorithms is mentioned and the only difference between k-means and EM is that in K-means the classification result give specific class or cluster while in the EM the result is given in a probabilistic way. In K-means the data points contributes to one of the given classes. If the classes are for example true or false, the points contributes either true or false. While in EM the points would contribute a little bit of their weights to the given classes.

The working process of Naive Bayes and Expection Maximization already seen in the theory section. The code below demonstrates how these works together by referring to the formula shown in the theory section. In Figure 3.1, Model-B is the ANN model that produced the best results among the supervised classifiers when predicting a system code for each unlabeled data set. The *SelfTraining Classifier* class from semi-supervised in scikit-learn was then chosen as the first choice to first shuffle and redo only 20% of the data set to labeled and the other assigned to -1. The implementation is as follows:

```
0
1 from sklearn import preprocessing
2 # for arragnging the proper index
3 def properIndex(dataFrame):
4     ind = []
5     for i in range(len(dataFrame)):
6         ind.append(i)
7     s = pd.Series(ind)
8     dataFrame.set_index(s, inplace=True)
9     return dataFrame
10
11 for i in range(len(unlabeled)):
12     res = ANN.predict([unlabeled.text[0]])[0]
13     unlabeled.system[i] = res
14 # Combining both the all the labeled and unlabeled, which is
     now labeled using ANN
15 frame_ = [labeled, unlabeled]
16 tot_ = pd.concat(frame_)
17 tot_ = shuffle(tot_)
18 # calling a properIndex to arrange the index
19 tot_ = properIndex(tot_)
20 le = preprocessing.LabelEncoder()
21 tot_['system'] = le.fit_transform(tot_.system.values)
22 X_, y_ = tot_.text, tot_.system
23 X_train, X_test, y_train, y_test = train_test_split(X_, y_)
```

```
24
25 # masking 20 percent of the y_train
26 y_mask = np.random.rand(len(y_train)) < 0.2
27
28 # X_20 and y_20 are the subset of the train dataset indicated
      by the mask
29 X_20, y_20 =
30     map(list, zip(*((x, y) for x, y, m in zip(X_train, y_train
      , y_mask) if m)))
31
32 # set the non-masked subset to be unlabeled
33 y_train[~y_mask] = -1
```

Listing 3.12: Code for initialising SelfTrainingClassifier

The implementation above increases the probability of predicting correct system code from the selfclassifier algorithm. This is because the neural network model had high accuracy in predicting the correct system code. The mixing part of the data set, right before assigning -1 to the 80 percent of the total data are also another helpful hint on predicting correct system code.

3.3 Results and Working Process of the System

During implementation, each system-code with more than 101 samples are chosen as labels. The samples and their system codes are distributed almost equally and looks as follows:

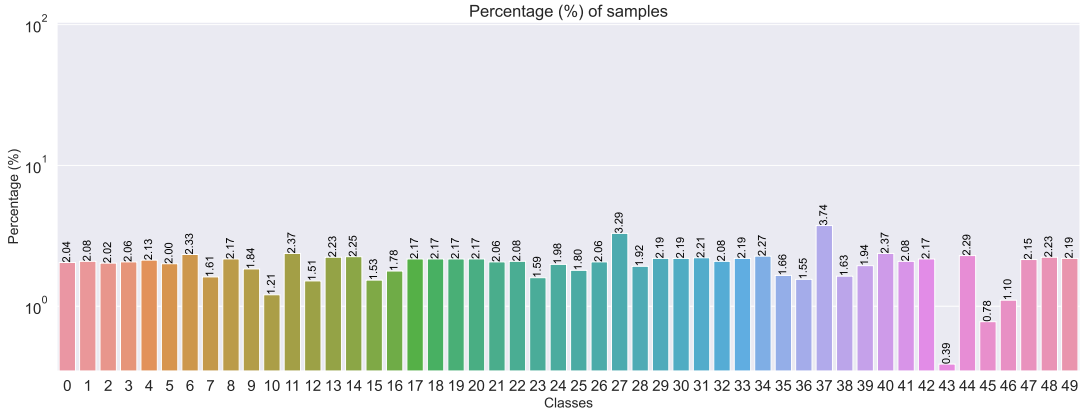


Figure 3.13: Top 50 labels

The traditional classifiers used in this project are Multinomial Naive Bayes, Linear Support Vector, Decision Tree, and Random-Forest, as stated in the introduction for ML section. And these are the first algorithms that have been tested in labeled data. The results of these algorithms, as well as other possible algorithms for classifying samples, are as follows:

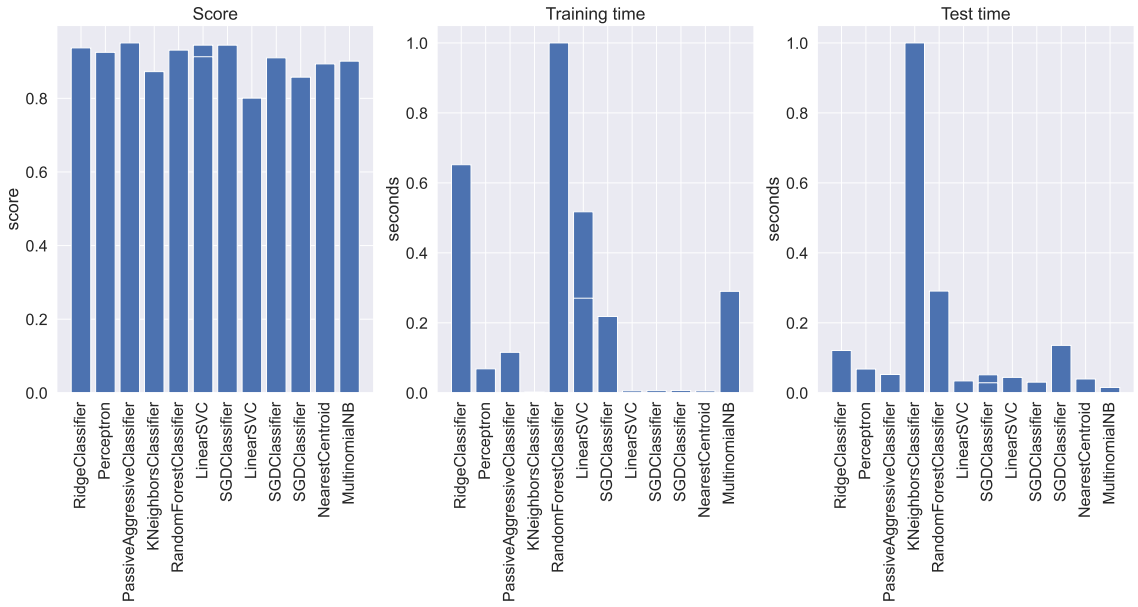


Figure 3.14: Result of traditional algorithms

The score for classifier algorithms is shown in the above result. The score takes into account not only the algorithms explained in the theory section, but also the algorithms that are related to the main algorithms. As it can be seen from the figure above, the result of the score lie around 85%. This is a good score because it covers the approximated result. One can also see the time spent in seconds, during training and testing. In the middle and right parts of the figure, there are some differences in time use. However, regardless of the time use, the accuracy remains constant. In some cases, the time spent on training by an algorithm is much greater than the time spent validating the data, and vice versa. Some observations will be made about this in the discussion sub-section.

In order to see most possible metrics performance from the algorithms, measures like precison, mse, r2_score and f1_score are taken by using sklearn. And the result of these measurments can one see in the figure below.

		Train time	Test time	Accuracy	Precision	MSE	r2_scor	f1-score	Dimensionality	Density
DecisionTree		1.681	0.032	0.892	0.892	6.410	0.913	0.892	---	---
RandomForest		4.974	0.050	0.922	0.922	5.100	0.931	0.922	---	---
Naive Bayes	MultinomialNB	0.029	0.005	0.910	0.91	2.5	0.96	0.91	43327	1
	BernoulliNB	0.029	0.019	0.858	858	9.217	0.875	0.858	43327	1
	ComplementNB	0.027	0.005	0.894	0.894	6.259	0.915	0.894	43327	1
SVM		7.913	2.119	0.903	0.903	4.428	0.940	0.903	---	---
KNeighbors Classifier		0.009	0.150	0.873	0.873	12.465	0.873	0.873	---	---
Ridg Classifier		3.149	0.012	0.937	0.937	2.369	0.968	0.937	43327	1
Perceptron		0.282	0.008	0.925	0.925	2.981	0.959	0.925	43327	0.105
Passive-Aggressive		0.597	0.006	0.945	0.945	1.742	0.976	0.945	43327	0.526
L1-Penality	LinearSVC(tol= 0.001)	1.433	0.004	0.913	0.913	5.849	0.920	0.913	43327	0.00
	SGDClassifier(max_iter=50, tol=0.001)	0.812	0.008	0.913	0.913	4.346	0.941	0.913	43327	0.002
L2-Penality	LinearSVC(tol= 0.001)	2.516	0.005	0.945	0.945	2.012	0.973	0.945	43327	1
	SGDClassifier(max_iter=50, tol=0.001)	0.299	0.004	0.952	0.952	1.892	0.974	0.952	43327	0.227
Elastic-Net Penalty		1.060	0.005	0.951	0.951	1.973	0.973	0.951	43327	0.034
LinearSVC (L1 based feature Selection)		1.381	0.004	0.901	0.901	8.412	0.886	0.901	---	---

Figure 3.15: Performances

The main classifiers and metrics discussed in the theory section are represented by the blue and red colors in the figure. Tolerence and various types of penalties are tested. Precisions for the algorithms (i.e., how many instances where the classifiers predict to be true are correctly classified as true) show a dominant higher value, indicating good performance among the classifiers. In terms of MSE, Naive Bayes (specifically MultinomialNB)

and SGDClassifier with l2 penalty perform better than the others. As a general rule, the lower the value and greater than zero MSE is preferred. So, if one were to use a traditional classifier, one of these two would be a better option. F1-score of all algorithms perform well, because their results are close to one. This means that the higher the precision and recall, the better the classifier performs - because it detects the majority of positive samples (high recall) while not detecting many samples that should not be detected (high precision). The formula from the theory section can also be used to verify it. It is important to note that "micro averaging" is used in all of these algorithms, which means that metrics are calculated globally by counting the total true positives, false negatives, and false positives [9].

In addition to traditional algorithms, three-layer Neural Networks were tested. This model was chosen again to visualize and quantify some of the above results. For example plotting the accuracy and loss versus epochs. In other words, to validate the results obtained by the algorithms described above. The NN training result was approximately 96%, and the validation result was approximately 92%. When compared to the results of the traditional classifications algorithm models shown above, this classifier produces little better results. This could be due to a variety of factors. For example, in this algorithm, important terms such as learning rate, epochs, batch size, and loss can be included. In addition, the visualization how the algorithm performs during training and validation of the data states by the figure below.

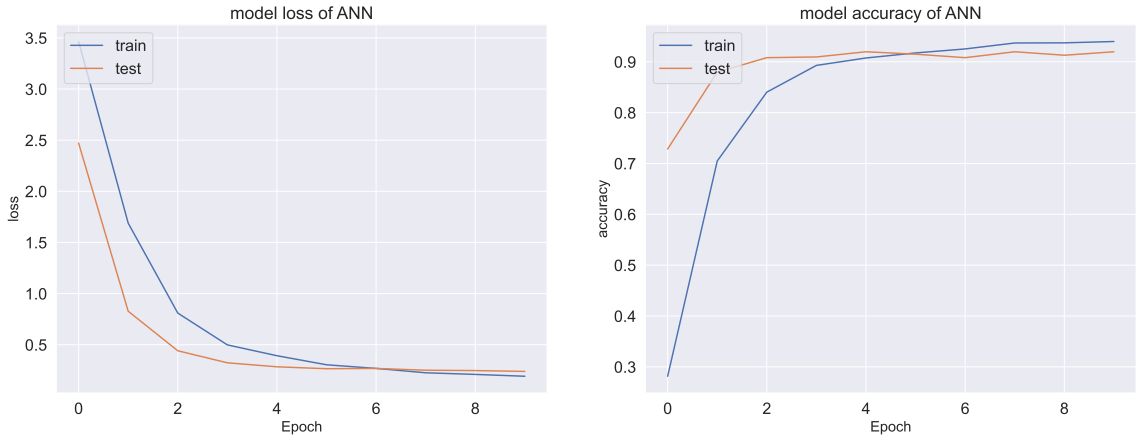


Figure 3.16: Accuracy and Loss vs Epochs 1

Overfitting is undesirable result where the model is trained well on the training dataset,

[9] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

but preformed bad in unseen/new data sets. If we see the figure above - loss vs epochs first, it tells as how the estimated result is far away from the actual result and when the line for the training and validation crossed to each other, it shows some overfitting. That means loss for the training data continues to decrease while the loss for the validation data decreases but slower than the training data set. In other word one can see the effect of this result on the right plot of the same row figure. The left plot shows the loss of the training and validation data set. It is also easy to see when the training accuracy become about 90% and then model performance changed - again overfitting, almost mirrored of left plot. However, model accuracy can be improved by adjusting the algorithm's hyper-parameters, i.e. the terms listed above. At the same time, ensure consistent accuracy by utilizing kernel regularizers found in the *Keras* library. The final result after adjustment was as follows:

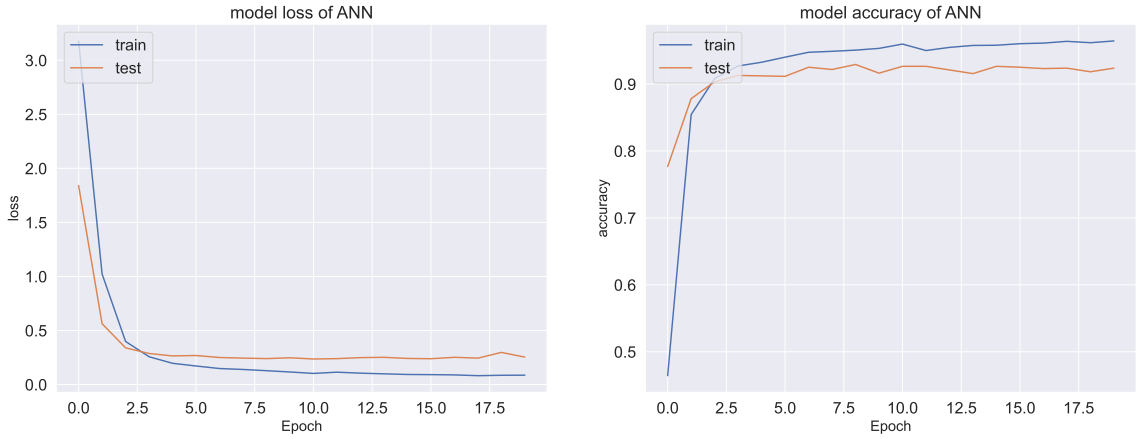


Figure 3.17: Accuracy and Loss vs Epochs 2

These results are achieved after several tests. It easy to see that the maximum result is almost constant after the training accuracy getting higher than the validation accuracy. Unlike the the previous figure the number of epochs in both figures are also different. In the previous figure the overfitting happen in early stage of epochs, while here the training and validation goes parallel. The loss in training and validation also getting closer during the 5 epochs. Thus, the accuracy of the final result 77 percent.

```
0
1 # initial parameteres and regulizers for adjustment
2 layers = 2
3 units = 40
4 dropout_rate = 0.2
5 reg = regularizers.l1(l1=1e-5)
```

```
bias_regularizer=regularizers.l2(1e-4)
activity_regularizer=regularizers.l2(1e-5)

# length expected output classes and input
top_20 = len(sys_to_num)
input_shape = x_train.shape[1:]

#expected output classes
out_units = top_20

# creation of the model
model = models.Sequential()

# Dropout layer to help prevent overfitting
# can include the initial parameters for improvment
model.add(Dense(units=units, activation='relu', input_shape=
    input_shape))
for _ in range(layers-1):
    model.add(Dropout(rate=dropout_rate))
    model.add(Dense(units=units, activation='relu' ))

model.add(Dropout(rate=dropout_rate))
model.add(Dense(units=out_units, activation='softmax'))
```

Listing 3.13: Code for Creation and Adjustment Neural Networks

The outcome of the semi-supervised part is determined by the outcome of the supervised part. Even the supervised part that participates in modeling the semi-supervised part is small, it is critical to have a model that can generate new data, as shown in 3.1. The Multinomial from the traditional type of machine learning as baseline and EM are chosen to perform the semi-supervised part of this project. The choice of Multinomial Naive Bayes is based on the strong result from 3.15, i.e good accuracy with low MSE. As it shown in the C section of 3.1, this model is the final step before the returning the class with it's belongs named-entities. The working process for each Naive-Bayes and Expectation Maximization can be found in the theory section separately. The working process for both these algorithm to preform sem-supervised algorithm looks as follows:

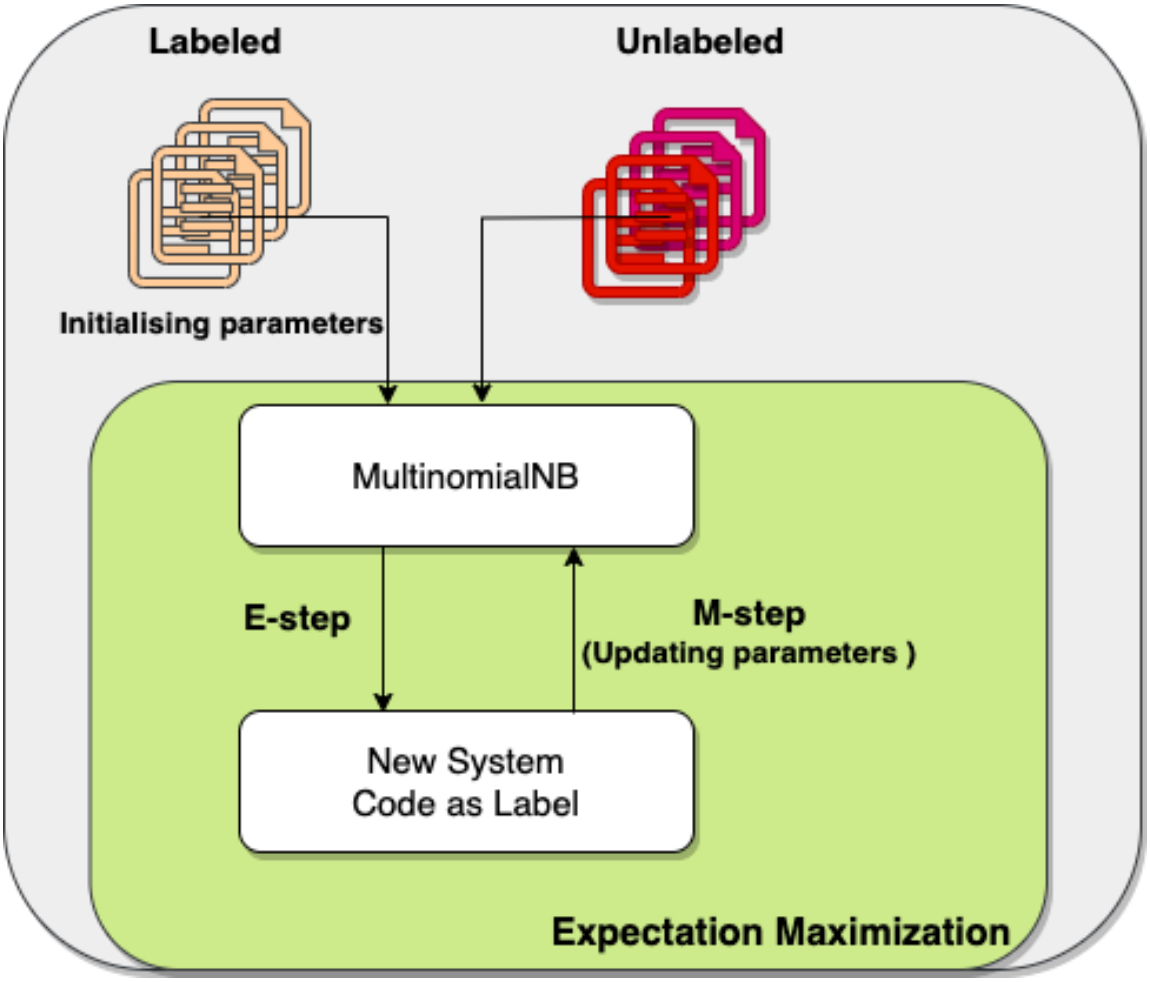


Figure 3.18: Process of Expectation Maximization

The above-shown figure produced 85% accuracy for the given initial 50 labels (system-code). This is expected, even if it cannot be verified due to the poor quality of the data; further reasoning can be found in the discussion section. However, in general, it can be reasoned that the MultinomialNB was used as a base-line in this process and has a good accuracy with law MSE, resulting in a good accuracy from the final step. Furthermore, to ensure the consistency of the results, cross validation on small sampled computed. Here is the figure that shows some measurements before arriving to the final result:

Labeled	Unlabeled	Ratio Labeled vs Unlabeled	NB-accuracy	EM-accuracy
1037	9337	0.1	0.67	0.21
3102	7240	0.3	0.77	0.21
5171	5171	0.5	0.81	0.25
7239	3103	0.7	0.81	0.36
8273	2069	0.8	0.82	0.5
9307	1037	0.9	0.82	0.57

Figure 3.19: Results with Max Iteration=30

A simple application is also created to ensure the working process of this project. The application includes the model from the final step (shown above), named entities operation, one html file, and other libraries to demonstrate the process from beginning to end. The application's front page looks like this:

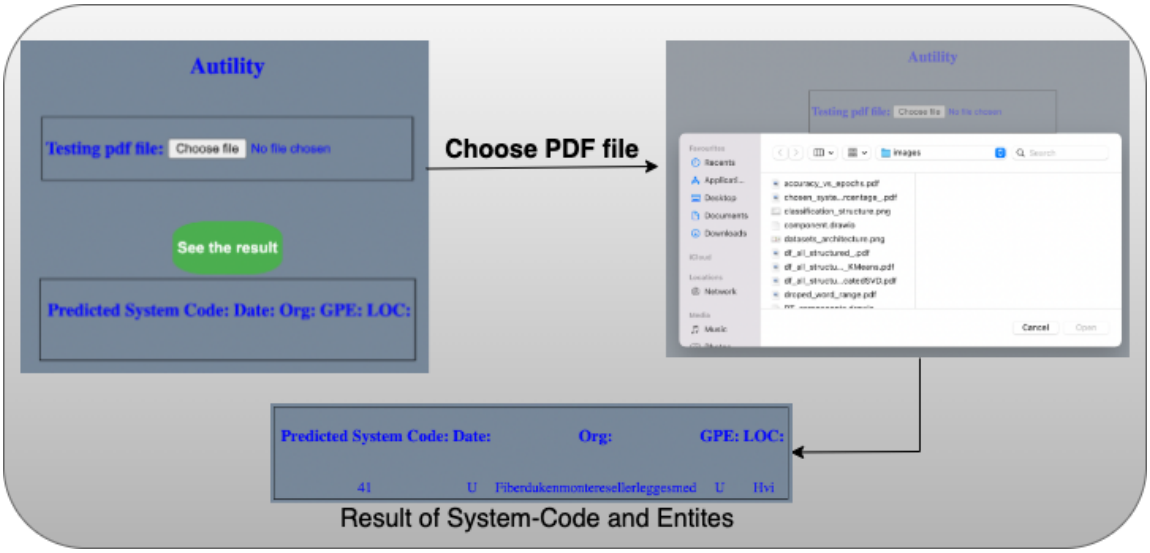


Figure 3.20: Application Working Process

The application takes a single PDF file and uses the saved model to categorize it according to its system-code. Furthermore, it extracts named entities, such as - Date, Org(Organization), GPE, and location. After pressing the *select result* button, all of these are displayed in the front page if they exist. In the example above the system-code and Org are displayed while the others are referred as U which stands for ‘Ùnknown”. All of the files and libraries necessary to run the application can be found in the attached zip.

Chapter 4

Summary

4.1 Discussion

The pdf files used in this thesis contain a variety of text types. The texts contain both words that may have meaning when learning the model and words that have no meaning at all. Due to lack of data, text augmentation was used during the implementation of the first part, which implemented the supervised part of machine learning. The augmentation is implemented by selecting random words from random samples that contain the same system-code. Then substitute a synonym for that word. This procedure is useful for categorizing the samples found in the total labeled dataset. The performances were also excellent. Even though the data shortage was resolved and the results were excellent, the data quality was poor when using the semi-supervised portion of the thesis. This is because, while the semi-supervised uses little data from the labeled part, the data must be related to the general data. This means that some files contain tables containing a large number of numbers that measure the physical components of buildings. That did not help learning any of the semi-supervised model. Because numbers are removed during preprocessing, the file is then left with only cm, mm, or kg, which have little or no impact on learning the model. In addition to that the texts are of single or double letters that do not give any information, the overall performance will be insufficient in terms of the final result. Another challenge in this project, some pdf files can also be unreadable, even the number of dataset show high, again this led to negative impact for learning the models. In some of the pdfs files can also be unreadable, which is negative side on having enough data.

The number of unique labels that exist in the structured part of data is much lower than the number of unique labels that given in the standard NS3451. This means The number of pdf files given in the structured data which again belong to the number of labels are very limited. By shrinking the number of labels that contain n number of files and augmented

the number files that already exist, it is managed to learn several type of ML models. The results gave an accuracy between 85% and 96% accuracy, but again the results applies for only 50 out of the 280 labels that exist in the standard NS3451. While the result achieved from the unstructured data can be see as a practical problem. This is because the files might belong to the labels that never seen below, that is labels that can only found in NS3451.

This can be a hint that is necessary to have a way where the model gather data continuously as stream and train the existed models in order to improve the accuracy and outcome quality.

4.2 Environmental accounts

Any task involving machine learning and artificial intelligence has an effect on our environment. Reflection on things from an ethical standpoint may impact the way the solution is generated and the working process in the future to some degree. Based on this, it is preferable to offer an explanation of ethical principles. The need for an ethical aspect may also be seen as a defender for many points of view on whether the project's generated part will have a beneficial or negative impact.

Before the start of this project, Autility and I had an open talk about the ethical implications of working on this project. This had a significant impact on how we worked with openness and responsibility for potential issues. Beyond that, it was clear that this was intended to enhance solutions to issues that increase problems of climate change. Even though there have been multiple instances where the development of ML and AI has had a negative influence, such as many jobs being threatened by AI and ML automation technologies, many people are optimistic that the rising role of AI will leave us better off [11]. Solving the climate change issue is a worldwide responsibility, and having this system solves not just the climate change problem, but also the time lost on classifying files. People who are doing this may also work on other projects that improve people's thought abilities. In general, a problem solved by ML should be met with a thorough analysis of the ethical considerations involved with it. This is because it is everyone's obligation to discover a solution that makes human work more efficient while not making the need for humans outdated. If this is the case, the development of automation generates more problems than it solves.

4.3 Conclusion

The point of the project was to create a system that used several NLP techniques and machine learning to extract information and categorize files in Autility based on the standard

NS3451. Solving a problem using NLP and ML, like in this case, is an opportunity to gain valuable experience in exploring and handling data. Despite the fact that the original data storage was never considered of as a process in this case, and the data quality was poor, it is managed to solve the problem.

Since a large portion of the work process happened during the Russo-Ukrainian conflict, which harmed the firm for which the thesis is written, I had to be adaptive in order to stick to the original plan. The unique experience obtained throughout the work process has improved my ability to engage with a solution-oriented viewpoint. During my first contact with Autility, several changes were made to the project concept. But, as mentioned multiple times throughout the discussion with Autility, the key objective has been to keep the solution being feasible. And the approaches for completing this assignment were chosen with the assumption that it is a project that would be expanded upon. Because we had certain constraints on the quantity and quality of data that I could access, this is a starting point for establishing realistic expectations for the outcome. The model's findings are restricted by the amount of the data, and there is a need for additional data, both in terms of the number of samples that represent the labels and the number of samples that represent individual labels. This might assist to represent a completed product.

Some of the system-code in the structured component has relatively few examples. This means that the model did not obtain enough experience with the samples that contained these labels to learn and identify them again. The text augmentation is conducted to enhance the data with system-code, so that the amount of data grows and the model to have exclusive classification obtains 89%. Having the same quantity of data for the other system-codes would result in a result that was near to the specified accuracy. This also applies to data of the same quality as the original data. With access to far bigger datasets, more complex models might be developed and perhaps reach even better accuracy. A simple neuron network model is included. However, because of the existing data, making it more complicated, such as adding additional layers, is unnecessary. The majority of the models utilized should be basic enough to handle bigger volumes of data.

Finally, this thesis is intended to act as a theoretical basis as well as a feasibility analysis on the topics that have been worked on in an acceptable manner. Based on multiclass filtering issues, the results provide a unique system code. It is also developed models for the structured component that provide 85% accuracy and various outcomes for the semi-supervised part depending on how the labeled and unlabeled are grouped, as shown in the figure 3.19. The output fulfills Autility AS's expectations and may be improved upon, and I'm fortunate enough to be able to point to a solution that is in accordance with the stated purpose.

Chapter VI

VI. Acronyms

AI	Artificial Intelligence
CART	Classification And Regression Trees
DL	Deep Learning
DTs	Decision Trees
EDA	Exploratory Data Analysis
EM	Expectation Maximization
IE	Information Extraction
ML	Machine Learning
MSE	Mean Squared Error
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NER	Named Entity Recognition
NN	Neuron Networks
POS	Part Of Speech
POS-T	Part Of Speech Tagging

GPE	Countries, cities, states
Org	Organization

VI. Tools used in this thesis

Figure VI.1: Python

Python is a programming language that is widely used in machine learning and deep learning.

Figure VI.2: Jupyter Notebook

Jupyter Notebook is an open source IDE that allows to create code in real time and share it with others. One of the key benefits of utilizing jupyter notebook or jupyter Lab, the next version of jupyter notebook, is the ability to execute snippet code.

Figure VI.3: Pandas

Pandas is an open source Python package for data manipulation and analysis created in the Python programming language. It includes data structures and algorithms for working with numeric tables, particularly multi-dimensional arrays. It is commonly utilized in data science and machine learning projects.

Figure VI.4: Matplotlib

Matplotlib is a library for plotting that works with the programming language Python and the NumPY extension for working with numbers.

Figure VI.5: ScikitLearn

Scikit-learn is a free and open source package that is based on NumPy, SciPy, and matplotlib. It includes techniques for classification, regression, dimensionality reduction, and other tasks.

Figure VI.6: Spacy

Spacy is an open source application created in the Python and Cython programming languages. It is used for sophisticated language processing across a variety of languages.

Figure VI.7: NLTK

Natural Language Toolkit is a collection of tools and applications for symbolic and static natural language processing of English texts developed in Python.

Figure VI.8: Tensorflow

Tensorflow is a Google-created open source platform. It is primarily intended for complex numerical computations and serves as the primary application for machine and deep learning.

Figure VI.9: Keras

Keras is an open source-code software that offers a Python interface for artificial neural networks. Keras serves as an interface for the TensorFlow library.

VI. Appendix

All code implementation is available here:
https://github.com/Tsegazab-Tesfay/TesMaster/blob/main/V9.ipynb
The setups and further information(README.md) about the thesis including code can be found here:
https://github.com/Tsegazab-Tesfay/TesMaster

Chapter VII

Bibliography

[1] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[2] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[4] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[5] Kurnia Muludi, Dwi H Widyantoro, Oerip S Santoso, et al. Multi-inductive learning approach for information extraction. In *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, pages 1–6. IEEE, 2011.

[6] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[7] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[8] Sunita Sarawagi. *Information extraction*. Now Publishers Inc, 2008.

[9] S. S Teri and I. A Musliman. Machine learning in big lidar data: A review. *International archives of the photogrammetry, remote sensing and spatial information sciences.*, XLII-4/W16:641–644, 2019.

[10] Maksim Tkachenko and Andrey Simanovsky. Named entity recognition: Exploring features. In *KONVENS*, pages 118–127, 2012.

[11] Weiyu Wang and Keng Siau. Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda. *Journal of Database Management (JDM)*, 30(1):61–79, 2019.