Machine Learning analysis of text in a Clinical Decision Support System

Dimitri Gharam
Abstract

Machine Learning analysis of text in a Clinical Decision Support System

Dimitri Gharam

Nurses at the Uppsala Emergency Medical Dispatch Center uses a computerized dispatcher system to prioritize patients calling the emergency number (112). The dispatchers at the emergency medical dispatch center register information into that system to help them determine the treatment necessary for the patient’s condition. One thing the nurses want to find out is whether a specific patient will require admission to the hospital. In addition to structured data from the decision support system, notes written by the dispatchers are documented. In this work, we have analysed ways we can use the text from ambulance dispatchers to predict outcomes using methods that enable computers to understand natural language called natural language processing, and have been implemented using machine learning approaches such as Classification and Deep learning developed in Python, SKLearn and Keras. To perform training using our data along with these approaches, we transformed our data using three types of representations: Bag-of-Words, TF*IDF and word vectors. The aim with these representations and approaches is for our machine learning models to be able to predict the likelihood of outcomes based on a given set of data. The results from the training gave us an understanding that some models performed better than the others, but also that the imbalance of the data prevented the models from generating more accurate results.
Preface

Ever since I was 15, my dream was to work with development of IT and now I have finally made it. This work has been chaotic with a lot of knowledge about technology I was not aware of, but it was important that I had to learn. There is a big difference both in learning and understanding where if you learn something and can imagine a similar moment with that newly acquired learning then you can implement it. After all these years of working with computer systems it is clear that you have to see it to believe it in order to understand something. I had to perform this thesis all by myself and with the correct guidance from my supervisor and reviewer, which I am very thankful for giving me the confidence to do this work and the excitement of implementing. I also want to thank everyone that helped me make this work possible, without your help, it wouldn’t be possible to reach a conclusion, also I want to thank my family for always pushing me to the end no matter what it takes.
# Contents

1 Introduction .................................................. 8  
   1.1 Purpose and goals ........................................ 9  
   1.2 Ethical aspects .......................................... 10  
   1.3 Delimitation ............................................. 10  
   1.4 Background ................................................ 11  
      1.4.1 Text mining and Natural Language Processing .......... 12  
      1.4.2 Analysis of NLP tasks ................................ 13  
      1.4.3 Analysis of NLP approaches .......................... 14  
      1.4.4 Summary of NLP tasks and approaches ................. 15  

2 Method .................................................... 17  
   2.1 Pre-processing ........................................... 17  
   2.2 Feature extraction ........................................ 17  
      2.2.1 Bag of Words ......................................... 18  
      2.2.2 TF*IDF ............................................... 18  
   2.3 Feature Engineering ....................................... 19  
   2.4 Word Embeddings .......................................... 19  
      2.4.1 Vector Representation ................................ 20  
      2.4.2 Word2Vec ............................................ 20  
      2.4.3 Character embeddings using Fasttext ................. 21  
   2.5 Introduction to Classification Methods ..................... 21  
      2.5.1 Logistic Regression .................................. 22  
      2.5.2 Naive Bayes .......................................... 24  
      2.5.3 Decision Tree ........................................ 25  
      2.5.4 Ensemble Methods .................................... 26  
      2.5.5 Evaluation of classification methods ................. 28  
   2.6 Neural Networks .................................... 29  
      2.6.1 Convolutional NN ..................................... 29  
      2.6.2 Recurrent NN ......................................... 30  
   2.7 Evaluation Metrics ........................................ 32  
   2.8 Implementation using Data Science Process ................. 34  
   2.9 Requirements Engineering ................................. 36  

3.5 Results from predictions .................................................. 57

4 Analysis ............................................................................ 59

4.1 Traditional Machine Learning approaches .................................. 60

4.2 Deep learning approaches ..................................................... 60

4.2.1 CNN and RNN comparisons .............................................. 61

5 Discussion ......................................................................... 62

5.1 Future Enhancements .......................................................... 66

5.2 Conclusion ....................................................................... 67

A Code architecture ................................................................ 68

B Plot results ........................................................................ 79

C Machine Leaning .................................................................. 86

C.1 Wellsprings of ML ............................................................... 86

C.2 Types of learning in ML ....................................................... 87

C.3 Deep Learning ................................................................. 87

C.3.1 Artificial Neural Network ............................................... 88

C.4 ML - Programming Languages ........................................... 90

C.4.1 Machine Learning APIs ................................................ 91

D Optimization of data ............................................................ 92

D.1 Binary vs Multiclass data ................................................... 92

D.2 Over- and underfitting ........................................................ 92

D.3 Hyperparameter optimization .............................................. 94

D.3.1 Hyperparameter tuning .................................................. 95

D.3.2 Explanation of hyperparameters ...................................... 96

D.4 Activation functions .......................................................... 99

D.5 Optimizer ....................................................................... 102

D.6 Search Algorithm .............................................................. 104
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An overview of data science</td>
<td>[3]</td>
</tr>
<tr>
<td>2</td>
<td>An overview of text mining approaches that can be used based on subject</td>
<td>[14]</td>
</tr>
<tr>
<td>3</td>
<td>Approaches of sentiment analysis</td>
<td>[19]</td>
</tr>
<tr>
<td>4</td>
<td>A visual of how the tf*idf frequencies are distributed</td>
<td>[28]</td>
</tr>
<tr>
<td>5</td>
<td>A visual example of CBOW vs Skipgram</td>
<td>[34]</td>
</tr>
<tr>
<td>6</td>
<td>Example of how classification works</td>
<td>[36]</td>
</tr>
<tr>
<td>7</td>
<td>Linear in blue vs logistic in orange</td>
<td>[37]</td>
</tr>
<tr>
<td>8</td>
<td>Logistic regression separating data</td>
<td>[38]</td>
</tr>
<tr>
<td>9</td>
<td>Gaussian Naive Bayes example</td>
<td>[40]</td>
</tr>
<tr>
<td>10</td>
<td>Example of a decision tree</td>
<td>[42]</td>
</tr>
<tr>
<td>11</td>
<td>Example of ensemble methods</td>
<td>[43]</td>
</tr>
<tr>
<td>12</td>
<td>Comparison of ensemble methods</td>
<td>[43]</td>
</tr>
<tr>
<td>13</td>
<td>Convolutional NN</td>
<td>[46]</td>
</tr>
<tr>
<td>14</td>
<td>Classic RNN architecture</td>
<td>[47]</td>
</tr>
<tr>
<td>15</td>
<td>LSTM to the left and GRU to the right</td>
<td>[48]</td>
</tr>
<tr>
<td>16</td>
<td>Bidirectional RNN in orange</td>
<td>[49]</td>
</tr>
<tr>
<td>17</td>
<td>Confusion matrix</td>
<td>[51]</td>
</tr>
<tr>
<td>18</td>
<td>Example of roc curves</td>
<td>[53]</td>
</tr>
<tr>
<td>19</td>
<td>A data science approach</td>
<td>[55]</td>
</tr>
<tr>
<td>20</td>
<td>Output data example</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Classification report example using dummy data</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Confusion matrix example using dummy data with true label on the y-axis and predicted labels on the x-axis</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ROC Curve from testing with real data to display our intention of applying ROC curves</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>The frequency distribution of the sentences</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>ROC curves on the decision tree</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>ROC curves on Naive Bayes Gaussian</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ROC curves on Logistic Regression</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>ROC curves on Ensemble</td>
<td></td>
</tr>
</tbody>
</table>
29 Classification report of LSTM and GRU, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s
30 Classification report of BiLSTM and BiGRU, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s
31 Classification report of CNN1 and CNN2, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s
32 Confusion matrix of LSTM and GRU
33 Classification report of BiLSTM and BiGRU
34 Classification report of CNN1 and CNN2
35 An example of how ANN is structured, source: [69]
36 The neuron calculation, source: [71]
37 Binary class vs multiclass, source: [38]
38 underfit vs good fit vs overfit, source: [80]
39 The learning rate will find the lowest loss value in the curve also known as the local minima, [86]
40 Multiple learning rates in a loss function, [89]
41 The Sigmoid function, source: [89]
42 The Tanh function, source: [89]
43 Multiple relus, source: [89]
44 SGD loss function, source: [89]
45 Grid search vs random search, source: [83]
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>NLU</td>
<td>Natural Language Understanding</td>
</tr>
<tr>
<td>DSS</td>
<td>Decision Support System</td>
</tr>
<tr>
<td>CDSS</td>
<td>Clinical DSS</td>
</tr>
<tr>
<td>EMD</td>
<td>Emergency Medical Dispatch</td>
</tr>
<tr>
<td>EMS</td>
<td>Emergency Medical Services</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>SL</td>
<td>Supervised Learning</td>
</tr>
<tr>
<td>UL</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest</td>
</tr>
<tr>
<td>EDA</td>
<td>Exploratory Data Analysis</td>
</tr>
</tbody>
</table>
1 Introduction

Uppsala University hospital uses a Clinical Decision Support System (or CDSS in short) that aid ambulance dispatchers to determine the amount of treatment needed for a specific injury or cause. It uses a finite set of questions whereof some of the questions have the ability to change the seriousness of the incident which makes this system a very important utility for the nurses when the patient arrives to the hospital. When an incident occurs, a person will call an emergency number; the recipient of the incident call will be the ambulance dispatchers at the Emergency Medical Dispatch (EMD) centers. The dispatchers will act as "the primary link between Emergency Medical Service (EMS) resources and the public" [1]. The dispatchers interact with the CDSS that aids the dispatcher to decide if the patient needs an ambulance with a given priority that ranges from life threatening to not life threatening.

Today, CDSS uses a knowledge based approach based on questions answered to determine the seriousness of an injury during the call, the dispatchers take notes that will justify the answered questions. Ideas were then conceived to create a solution that will take these notes and apply a machine learning approach to predict if the patient requires to be admitted to the hospital or should be referred to a local health care center.

This work is a collaboration with the hospital, that analyzes the possibilities to use texts as inputs and evaluate the predictive result of various machine learning models to predict hospital admission. This work will focus on areas of Machine Learning and Natural Language Processing that will generate risk assessments to support nurses to provide medical decisions as well as a graphical representation of the data. These assessments and graphical representations of the data will represent a result where it describes how the machine learning model have trained using the text and then how it predicted the assessments, and these results are comparable to the actual assessment generated by the dispatcher to give an understanding in how the model performed during the predictions.

\footnotetext[1]{This report will contain a lot of abbreviations, all of them will be defined in the list of abbreviations}
1.1 Purpose and goals

The knowledge-based approach generates a decision based on what questions have been answered. The text that has been written by the dispatcher should also be taken into consideration since it also contains vital information (about the cause). The goal of this project is to create a solution that takes these texts and predict an outcome; that outcome can vary from either yes or no to a decision out of a finite amount of possible answers.

One of the requirements given, is that the solution has to be independent from the CDSS system so that the solution doesn’t need integration to it. The next requirement is that the method to analyse the text has to be based on machine learning. One method is to apply Natural Language Processing (NLP) to analyze the text and then using that data to generate a desirable result. The final requirement is that we will perform analysis of the text by applying the Swedish language.

The purpose of this project is to develop a software solution that takes text as input and generates an outcome as an output and have to use machine learning as a base requirement. A comparison between multiple approaches is required, in order to find out which one is most suitable for our solution; therefore studying these approaches in terms of structure and functionality and what kind of methods can be used to evaluate these approaches.

The goals of this project is to:

1. Survey the literature and the internet and consult experts to decide a feasible approach to analyze text.

2. Decide how can we utilize the extracted words (with the knowledge from aim 1) to generate a decision.

3. Implement algorithms so that it can be trained on the data that is given by the user in a machine learning model to predict likelihoods e.g. hospital admissions.

4. Test, evaluate and optimize the implemented models in order to create an acceptable solution.
1.2 Ethical aspects

Since this project is a collaboration with the University Hospital, some principles have been taken into consideration to ensure that this work does not violate any ethical rules or personal integrity. The work was performed without access to identifiable patient data including raw free-text notes. Software was developed by the author, and acceptance testing was performed on actual data by hospital employees. We will not in this work present any form data that is connected to physical entities. The solution was implemented using open source code from various sources and will therefore be available for others to download.

1.3 Delimitation

Since this work focuses on Natural Language Processing where we generate an output from text, methods like speech recognition is excluded of this work. We will try and exclude irrelevant methods that aren’t useful for our project. Also, worth mentioning, is that the CDSS is used in a few hospital. Our work will be consisting of applying data science. Data science is, according to [2], a "study of where information comes from, what it represents and how it can be turned into a valuable resource". It is used to apply mining approaches for both structured and unstructured data to "identify patterns that can help an organization rein in costs, increase efficiencies, recognize new market opportunities and increase the organization’s competitive advantage"[2]. The areas that are covered by data science is according to [2]: mathematics, statistics and "computer science disciplines" including techniques such as machine learning, data mining and visualization, which is suitable for our work.
Our organization is the University Hospital and our goal is to analyze data from text in order to find relations between text and hospital admissions meaning that data science is eligible for this work. It will delimit our work and exclude unnecessary approaches. By applying data science, we can according to [2] "interpret, convert and summarize the data" that is collected and processed in order for the data to be "useful" to be implemented in machine learning approaches.

1.4 Background

Text is a "human-readable sequence of characters"[4]; basically a collection of sentences that can be considered to be a collection of unstructured data according to [5]. To explain it in context: we have data that we need to extract information from and then based on that information, we can extract knowledge out of this information [5], i.e. we need to perform an analysis which we can call text analysis [6].

Text analysis is a method that "parses text in order to extract machine-readable facts from them" [6], and its purpose is to "create structured data out of free text content" [6]. The term text analysis can according to [6] also be referred to as Text Mining, and according to [7], it is similar "in nature" to data mining but focuses on text rather than "structured forms of data"[7].
In [8], text mining, is a combination of both Statistical NLP\(^2\) and Data Mining\(^3\). One challenge when dealing with text mining is so called text ambiguities. In [9], when interpreting texts, the outcome of the interpretation can have multiple outcomes, i.e. one interpretation of one sentence can generate an output that differs from the human understanding of the sentence. One solution to this issue is to examine other "portions of the text"\([9]\) i.e. analyze the other sentences in the text to create a context of what it actually is describing about.

In order to make this work understandable and relatable, other works that have been performed can be mentioned here. E.g. [10] has had on a similar approach where it tries to answer if NLP can be used in the CDSS using multiple theories. Another work is to use text mining and NLP [11] in a similar area using concepts like information extraction, information retrieval, categorization and pattern matching. Melton published in the Journal of the American Medical Informatics Association [12] about how to use NLP and data collection to find traces of Discharge diagnosis and a patient’s medical records. One more approximately similar work is performed in [13] where they analyze text to extract data to generate a severity level on a certain damage caused by the patient. The practical parts of this work is a variation of sentiment analysis that is used to predict outcomes based on texts, of which there are multiple implementations and variations available on the web in order to take inspiration from and validate our work to theirs.

1.4.1 Text mining and Natural Language Processing

The Venn diagram (figure 2) from [14], describes what kind of approaches can be used based on what kind of work is required. E.g. assume there is a dataset of text stored in a database and the goal is to retrieve information based on that text, then the most likely approach to use is Information Retrieval\(^4\). For this work, the goal of this work is to predict admissions to the hospital based on the texts, which means one required area is statistics. Another goal is to take text as input, along with the order of the words in the sentence and the language in which the order is structured for analysis, but also, let the computer perform all the analysis, which means applying the area of computational linguistics. Finally, the software in which the analysis will take place, has to learn itself in order to improve its prediction, which means another focus

:\(^2\)which is a "set of algorithms for converting unstructured text into structured data object"\([8]\)

:\(^3\)which has a set of "quantitative methods that analyzes these data objects to discover knowledge"\([8]\)

:\(^4\)See figure 2
area is AI and Machine Learning. By looking at figure 2, only Natural Language Processing (or NLP), is most suitable, based on the requirements defined in the aims section.

Liddy [15], defines NLP as a "theoretically motivated range of computational techniques, for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis, for the purpose of achieving human-like language processing for a range of tasks or applications".

![Figure 2: An overview of text mining approaches that can be used based on subject, source: [14]](image)

1.4.2 Analysis of NLP tasks

In [16], they describe three NLP tasks along with their representations, goals and applications:

1. Text Classification:
   - Representation: Bag-of-Words
   - Goal: Predict tags, categories and Sentiment

---

\(^5\)actually five but chose three for this work  
\(^6\)TF*IDF can also be used for text classification
• Application: Filtering spam emails and classifying documents based on dominant content

2. Word Sequence

• Representation: Sequence of words
• Goal: language modeling, predicting previous or upcoming words, text generation
• Application: translation, chat bots and predict POS tags for each word in sequence and Named Entity Recognition (NER)

3. Text Meaning

• Representation: Word Vectors and the mapping of words to vectors (n-dimensional numeric vectors) aka embeddings
• Goal: to represent meanings of texts
• Application: Finding similar words (similar vectors), Sentence Embeddings, Topic Modeling, Search and Question Answering (QA)

1.4.3 Analysis of NLP approaches

Along with the tasks mentioned earlier, there are certain approaches that can be used to apply these tasks, and according to [16], there are three tasks:

1. Rule-Based - This is the traditional approach. It uses a "hand-crafted system of rules"[17] that is based on the same structure that humans builds up grammar. A rule-based system is based on a set of grammatical rules and improves with translations and synonyms. It doesn’t require large scales of input data but requires a lot of expertise to structure each component to create these rules.

2. "Traditional" Machine Learning - According to [16], traditional describe methods such as "probabilistic modeling, likelihood maximization, and linear classifiers". This approach doesn’t implements deep learning since the structure resembles a neural network with one hidden layer[7]. This approach uses components such as: training data (corpus), "feature engineering"[8] as input and a model to predict based on the input data.

---

7See appendix D
8Feature engineering is applied on features originated from the process of extracted features (in this case our
3. Neural Network/Deep Learning - Applies a Neural Network model, it doesn’t require feature engineering, but it uses a "vector representation of words"\(^9\)\(^\text{[16]}\). To transform our text, we need a "large corpus" where large text has processed word vectors using word vectorization methods\(^10\). When it comes to Neural Networks, \(^\text{[16]}\) mentions that either Convolutional Neural Networks or Recurrent Neural Networks can be used.

### 1.4.4 Summary of NLP tasks and approaches

The tasks and approaches that have been defined for this work are either: text classification, word sequence and text meaning, along with our desired approaches such as the traditional machine learning and deep learning. For our work, based on the mentioned tasks, text classification is considered to be more relevant than the other tasks, since we need to generate a decision based on the text if a patient needs to be admitted to the hospital or not, i.e. answer yes or no if the patient needs to be admitted\(^11\). Yes and no answers are binary subjective information, and it can be generalized into multiple answers and these answers are multiple parts of subjective information. These answers can be extracted and analyzed from data. Therefore the choice of task for this work will be sentiment analysis along with machine learning approaches such as classification\(^12\) and deep learning.

Sentiment analysis is a method that is able to analyze people's "opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes"\(^13\)\(^\text{[18]}\). By using text as input we can generate a output that can describe based on a given text a specific verdict\(^14\).

---

\(^9\) Raw text transformed into numbers that can be inserted into the Neural Network

\(^10\) such as Word2Vec or FastText

\(^11\) The text will contain data that will answer that question which cancel out the option for predicting tags or categories due to irrelevancy, and sentiment is a close approximation of the desired task we want to perform our work

\(^12\) Supervised Learning, see appendix C

\(^13\) There are also other types of approaches of sentiment analysis that focus on other aspects and areas such as "opinion mining, opinion extraction, sentiment mining, subjectivity analysis, affect analysis, emotion analysis, review mining, etc."\(^\text{[18]}\)

\(^14\) This approach can be considered as Supervised Learning since you have text and a "verdict"\(^\text{[18]}\) where you can use methods that will enable machine learning models to predict new verdicts on newer texts, based on the data that it is trained on
E.g. Suppose that there is a table with 3 columns where one column is the text, and the other 2 columns are sentiments, which we can call "admission", and "breathing". The values that are feasible for these sentiments are: "yes" and no. Each text has been given a sentiment by the ambulance dispatchers, and the goal is to apply sentiment analysis, to create a classification model that takes a couple of rows as training data where the text is input data and the sentiment is output data, which then will take the rest of the rows that haven’t been used for training, to be used as testing where we will compare the predictions generated from the model and compare them to the actual sentiments that were defined by the ambulance dispatchers.

In the method section, we will introduce the theories mentioned in figure 3 for the supervised learning approaches of sentiment analysis and also how the text can be transformed into data that will be inserted into our machine learning models using preprocessing and text encoding. We will introduce text encoding methods such as Feature extraction, feature engineering and vector representation.

Figure 3: Approaches of sentiment analysis source: [19]

---

15 We are omitting primary keys
16 We will exclude Rule-based classifiers, Bayesian Networks and Maximum Entropy since it will complicate our work
17 which are Bag of words and TF*IDF
18 a compliment for feature extraction
19 since we want to cover how our text can be transformed into data for our models
2 Method

Since this section is large, it will first present the theories that has been implemented, and then present the practical approaches that has been used along with applying the data science process, which is the way of working in order to reach a result.

2.1 Pre-processing

The reason why we need to preprocess the data according to [20] is that data extracted from the "real world" is "often incomplete, inconsistent and/or lacking in certain behaviors" which can generate a lot of "errors". Traditional data preprocessing is a 5 step process [20] that includes cleaning, integration, transformation, reduction and discretization. Having consulted Joakim Nivre which is an expert in NLP, he mentioned methods for preprocessing such as tokenization [20] and lower casing [21]. The tokenization can extend to not only generate words, but also to N-grams, which are "substrings of fixed length N" [22] where the text can be divided into chunks with a fixed length.

2.2 Feature extraction

When we have performed preprocessing on our text, we want to extract the features of our text to generate data to be used as inputs. There are a couple of feature extraction methods: Bag of Words and TF*IDF which are described below. In [23] they mention that Statistical NLP was the primary method of handling NLP tasks, however, they also note that this approach suffered from the "curse of dimensionality" [23] One way to prevent this is to reduce the number of dimensions, and in this case, we can apply a method called "Distributed representations" [23]. It means "a many-to-many relationship between two types of representations" [24] and there are two important embedding methods that are appropriate: word embeddings and phrase embeddings.

---

20 which "convert sentences to words" [21]
21 capital "A" and lower case "a" have different values when they are transformed to numbers despite being similar in terms of letters. Removing special characters such as periods and commas is also essential for this work
22 Substrings are an ambiguous definition where it can be referring to a series of characters or a series of words
23 occurs when there is too much data for the machine learning model to find patterns (e.g. going from 3D to 4D or even higher) that can in this case bind words together to form a sentence
24 distributed representations are also called "distributed embeddings" [23]
2.2.1 Bag of Words

Bag-of-Words (BoW) is defined as a "representation of text that describes the occurrence of words within a document" [25]. The purpose of being defined as a "bag" of words according to [25] is that it doesn’t preserve structure in the sentences. Another approach of using BoW is to store N-grams of words and count the occurrence of these N-grams. The downside of applying BoW is:

1. The bag can expand with both words and N-grams

2. As the bag expands, it will become time consuming if we have a large amount of text where we have to count how many times that word have occurred in that sentence

3. Since BoW just counts how many times words/N-grams have occurred in the sentences, it doesn’t preserve order of how the words constructed the sentence from where they have been extracted.

4. BoW doesn’t "capture information about a word’s meaning or context" [26] [26].

2.2.2 TF*IDF

In regards to Bag of Words where we calculate the number of words that occurs in a text, Term Frequency multiply Inverse Document Frequency (TF*IDF) is used in "information retrieval to represent how important a specific word or phrase is to a given document" [27], which is another encoding method used for enabling text classification. The reason why TF*IDF is interesting is that it tries "to make sense of a population of unstructured content to score what it’s about" and "how strongly it represents that topic or concept versus other documents in the sample population" [27] [27]. In conclusion TF*IDF counts the frequency of the words rather than the occurrence of the words.

---

25 It counts the number of times that word has occurred in the document.
26 i.e. we don’t get the semantic representations of these words.
27 In other words if we have free text without any type of syntactic or semantic rules, we can use TF*IDF to calculate which words are used frequently for what context that text is used.
2.3 Feature Engineering

According to [27] TF*IDF can be extended using a method called Latent Semantic Indexing or LSI to "rank documents based on relevance against a specific term or topic" [27]. LSI is used as a method to "produce low dimensional representations using word co-occurrence" [29]. To enable LSI, we can use a method called Single Value Decomposition (SVD) [30] to "identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text" [30]. In conclusion, LSI can be an important component for TF*IDF [28] that focuses on "words that are used in the same contexts tend to have similar meanings" [30].

2.4 Word Embeddings

Word embeddings is one of the three tasks mentioned in NLP used for text meaning. It uses vector representations of texts to parse it to a machine learning model that predicts either word similarities or for question answering (QA). There are many solutions that uses word embeddings but the two most popular methods according to [31] are Word2Vec [29] and FastText [30].

---

28 we will try and use this also in bag of words
29 made by Google
30 made by Facebook
31 GloVe is also mentioned in [31], but didn’t satisfy our requirement number 3
2.4.1 Vector Representation

Word vector is basically a "row of real valued numbers where each point captures a dimension of the word’s meaning and where semantically similar words have similar vectors" [26]. With this definition, we can get a good connection of words to another word e.g. "nose" and "sneeze" can give us "allergies" and if we add the words "hot" and "forehead" we will get "fever" and associating these words with a person, we can generate a semantic definition of "that person has got a cold". Word vectors originates from two sources of calculations according to [26]: the "counts of word/context co-occurrences" and "predictions of context given words". Vector representation can be presented as either a one dimensional vector for each sentence, or a multi-dimensional matrix for the complete document as input for our models.

2.4.2 Word2Vec

In [32] they mention that Tomas Mikolov introduced methods that "revolutionized" word embeddings such as Continuous Bag-of-words (CBOW) and Skip-grams, they go under the common name: "Word2Vec"[33]. CBOW tries to predict given a set of words, which word to fill in to generate what word can be inserted into the set. Skip-gram takes one word from a sentence and calculates the probabilities of the words surrounding the given word (see figure 5). One challenge these models can face is "Polysemy"[33] which means that one word can have multiple meanings, and one approach they propose to prevent it is to include multilingual parallel data that introduce "multi-sense word embeddings"[33], which when translated generates multiple meanings of those words.

---

32 such as CBOW and skip-grams which we will come back to
33 i.e. anyone that uses a method of word embedding mentions Word2Vec, has to choose to use either CBOW or skip-grams in their approach
2.4.3 Character embeddings using Fasttext

Character embeddings are more advanced embedding methods. Where word embeddings focus on the relationship of words, character embeddings focuses on the relationship between the characters. This approach can be very useful in eastern languages such as Chinese [23], and building words at a character level can help avoid events such as "word segmentation" and "Out-Of-Vocabulary" situations [23]. Recently, according to [23], character embedding methods have been considered to be more "interesting" than word embeddings. For example, one researcher found a way to improve the representation of words by using multiple characters in morphologically-rich languages and used a skip-gram method that represented words as "bag-of-character n-grams" [23], which was faster[34] than word embeddings and allowed "training models on large corpora quickly" [23] thus creating the API Fasttext.

2.5 Introduction to Classification Methods

Classification is an approach used in supervised learning to organize data based on their labels, and is one of two selected NLP tasks. The reason of applying Supervised Learning is that it "learns from labeled data" [35] and after the algorithm has trained from the data, it can be able to determine "which label should be given to new data based on pattern and associating the patterns to the unlabeled new data" [35] where the data in this case is the text. There two types of SL approaches according to [35]: Classification and Regression. Classification "predicts the a category the data belongs to" [35] and it is "used for predicting discrete responses" [35] where you can apply e.g. "Spam Detection, Churn Prediction, Sentiment Analysis, Dog Breed..."
Detection" [35], and Regression "predicts a numerical value based on previous observed data" [35] and you can use it for e.g. "House Price Prediction, Stock Price Prediction, Height-Weight Prediction" [35]. There are multiple algorithms that can be used in Supervised Learning.

If we look at the figure 6, in the left plot, we can see two types of data, which we can call class 1 and class 2, and a red line that will act as a discriminant that will separate the 2 classes. By applying a classification method, that discriminant will be able to adapt to distinguish the 2 classes apart which we can see on the right plot.

Figure 6: Example of how classification works, source: [36]

Worth mentioning that the theories of these algorithms mentioned here are for the most part from [35].

### 2.5.1 Logistic Regression

There are two types of "Regression" methods: Linear and Logistic regression. Logistic differs from linear where it generates a binary non-numerical output like a 0/1 output and a discriminant between these outputs. It works according to [35] by first applying linear regression where the threshold can be assumed as 0.5. The output of the result is inserted into a logistic Sigmoid function where the purpose is "to get the probabilities" [35] of the data belonging in either one of the outputs. The final result is the "logarithm of the probability of the event occurring to
the logarithm of the probability of it not occurring"[35].

In figure 7, we can see how both the linear regression and logistic regression behaves; both the functions are obviously continuous where the linear function

\[ y = b_0 + b_1 x \]

increases linearly (converges) towards infinity while logistic regression uses

\[ p = \frac{1}{1 + e^{-(b_0 + b_1 x)}} \]

where the function converges to 1 when x becomes larger which is why logistic regression performs better than linear. In the image below is a demonstration where the data from two different categories are separated by the logistic function.

Figure 7: Linear in blue vs logistic in orange, source: [37]

Figure 8: logistic regression separating data, source: [35]
2.5.2 Naive Bayes

Naive Bayes classifier originates from the "Bayes' theorem with the independence assumptions between predictors, i.e. it assumes the "presence of a feature in a class is unrelated to any other feature" [35]. Naive Bayes is built up according to [39] as a "a family of probabilistic algorithms that take advantage of probability theory and Bayes' Theorem". The Naive part of Naive Bayes is described in [39] as if we have a sentence, we are not looking at the whole sentence to classify a label, but rather at the words that build up that sentence given a certain label. In other words, each word given in the sentence, their probability given a certain label is being calculated, and the final product of those probabilities is the final probability given the label no matter the order of the words [39].

![Gaussian Naive Bayes example](image)

Figure 9: Gaussian Naive Bayes example, source: [40]

In [35], they describe the components of the equation in the Bayes Theorem:

- **P(class)**: describes the probability of the class
- **P(data)**: describes the probability of the "predictor or marginal likelihood"
- **P(data|class)**: describes the probability of "the likelihood which is the probability of predictor given class".
- **P(class|data)**: describes the probability of "class (target) given predictor (attribute). The probability of a data point having either class, given the data point. This is the value that we are looking to calculate."

\[^{35}\text{In this case we have a set of text and a label and we want to calculate the probability of a given text to a label}\]
\[^{36}\text{if we for example inserted another word into that sentence, the final probability will change}\]
The probability can be calculated by the following steps:

1. **Calculate Prior Probability**, calculate $P(\text{class})$
2. **Calculate Marginal Likelihood**, calculate $P(\text{data})$
3. **Calculate Likelihood**, calculate $P(\text{data|class})$
4. **Posterior Probability for each Class**, calculate $P(\text{class|data})$
5. **Classification**, calculate the data and which classes they belong to in terms of a probability threshold; when a new data is inserted into the model, given its data we can calculate based on the probability where that data should belong.

There are three types of Naive Bayes methods: Gaussian, Multinomial and Bernoulli. Gaussian is based on the binomial/normal distribution of continuous data. The difference between Gaussian, Multinomial and Bernoulli according to [41], is that Gaussian works best with continuous output data\(^{37}\), Bernoulli works best with Binary output data and multinomial works best with discrete data\(^{38}\).

### 2.5.3 Decision Tree

Decision trees are designed to specifically be used for "decision and decision making"\(^ {42}\) and are constructed as a upside down tree\(^ {39}\). The reason for this design is that, if the leaf is a sub-level root it can extend the tree until it reaches a leaf, which will be then the final decision. A decision tree can have multiple leafs based on the type of data: binary or multiclass. Figure 11 shows an example of a binary decision tree that can be configured to create a tree with two outcomes in each branch. If we look at the example, we can see the root and two branches where it connects to either an leaf or another branch and the algorithm will stop at the leaf.

---

\(^{37}\) data that changes over time

\(^{38}\) Categorical data

\(^{39}\) where the root (the discriminants) at the top and the leaves (the class) below the root (see the figure 11)
2.5.4 Ensemble Methods

Ensemble model is defined as a "team of models" and "when several models are trained separately then vote or are averaged to produce a prediction" [35]. In other words: we can train multiple supervised algorithms individually and then merge them to achieve a better prediction. There are two types of ensemble methods: Random Forest and Gradient Boosting. Random Forest (or RF in short) is based on bootstrap aggregating, or bagging, where the data is used by multiple models such as e.g. SVM, Naive Bayes and Decision trees. When the different models have been created and made their predictions, a vote will be used to reach a final prediction based on what model perform best [35].

Figure 10: Example of a decision tree, source: [42]

Figure 11: Example of ensemble methods, source: [43]
Gradient Boosting is a "strategy that trains a series of weak models, each one attempting to correctly predict the observations the previous model got wrong" [35]. It creates predictors in sequence that takes the output of the previous prediction as an input to the next model. The procedures of Boosting are described based on [35]:

1. Initialize predictions with a simple decision tree
2. Calculate residual[^40] value
3. Build another shallow decision tree that predicts residual based on all the independent values
4. Update the original prediction with the new prediction multiplied by learning rate
5. Repeat steps 2 through 4 for a certain number of iterations (the number of iterations will be the number of trees).

The reason for choosing RF is that it trains models in parallel, it performs "unweighted voting" [35] for the final prediction and it is simple to tune but difficult to overfit. The reason for choosing GB is that it trains the models in sequence, uses "weighted voting" for the final prediction and it is harder to tune and easy to overfit.

[^40]: the actual value subtracted by the predicted value

Figure 12: Comparison of ensemble methods, source: [43]
2.5.5 Evaluation of classification methods

In this section, we will evaluate the classification algorithms based on strengths and weaknesses to ensure how we will enact to implement these algorithms. Note that we will also use materials from [44]. For optimizations of classification models (or machine learning models in general) are described in appendix E.

Logistic Regression:

- **Pros**: Easy to regularize to prevent overfitting (will come back later for this).
- **Cons**: This is a part of the probability classifier and in [44] the model "tends to under perform when there are multiple or non-linear decision boundaries".

Naive Bayes:

- **Pros**: It is computationally faster than SVM
- **Cons**: Choosing the right Naive Bayes is tricky since we have 3 types of Naive Bayes: Gaussian, Multinomial and Bernoulli, where Gaussian and Bernoulli are suitable for binary classifications while multinomial is for multiclass classification.

Random Forest:

- **Pros**: Can reduce overfitting by "averaging several trees", i.e. the classification models used in random forest [44]
- **Cons**: Computationally expensive (takes a long time) and difficulties occurs in visualizing the model "or understand why it predicted something." [44]

Gradient Boosting:

- **Pros**: It can perform better than Random Forest according to [44] and have more hyper-parameters than RF to improve the model.
- **Cons**: For each model included in the Gradient Boosting algorithm it will take longer time to train everything, since the algorithm will build the models "sequentially" [44].

Decision Trees:

- **Pros**: Easy to model since it can create leafs and subroots based on the labels; simplifies the visualization to enable the developer to understand how to model the tree.
- Are able to "handle numerical and categorical data" [12]
- **Cons**: If the labels are unbalanced, the tree will have leaves that are more biased.
- Decision trees can create high variances that can result in overfitting which means that the slightest changes in the data can create a "completely different decision tree" [12]. We will talk about overfitting in a later section.

The conclusion here is that we need to train models using these algorithms and evaluate in the result section both the result and also using the pros and cons of these models, to give us the best working model.

### 2.6 Neural Networks

Neural Networks or "NN" are the second approach of NLP that will be used in our work. There are 2 NNs that are suitable for this approach: CNN and RNN. We will describe how we can optimize our NNs to perform faster or better in Appendix E.

#### 2.6.1 Convolutional NN

In [45] a traditional CNN consists of a convolution section, pooling section for the convoluted data and "fully connected" layers, which is a classic neural network. In [23] before entering the convolution layer, the sentences are transformed to vectors by applying word embeddings. One application for CNN is to apply more hidden layers to the MLP, thus creating Deep CNN and can be used for text summarization. But there are possibilities to tweak the CNN to increase the importance of pooling and apply HMMs for e.g. speech recognition [23].
In figure 14, CNN is described how data in the form of a matrix is using convolution and pooling to then predict a result. Our input matrix will be convoluted into convolutional kernels of a specific size and then these kernels will extract a desired value using pooling; pooling can be used in 2 ways: Max-Pooling and Average-Pooling. Max-pooling extracts the maximum value of that kernel and average-pooling takes only the average value from each kernel.

2.6.2 Recurrent NN

A traditional neural network that starts at the input layer, parses through the hidden layer(s) and reaches the output layer are called feed-forward NNs (FFNNs); NNs that can go backwards are called Feedback NN or FFNNs with back-propagation.

In figure 15, we can see how data generated from one hidden layer is transferred to the previ-
ous hidden layer and therefore improve the result generated in that specific layer, thus giving the NN "memory from previous computation" [23]. It doesn’t sub-sample the data but instead takes the output data as input data. One problem with this architecture is that the activation function of RNN can suffer from a "Vanishing gradient problem" [23]. To prevent this problem, two variants of RNN can be used: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) [23]. LSTMs have additional "forget gates" that makes it overcome the VGP and perform back-propagation at an "infinite number of time steps" [23]. GRU is less "complex" than the LSTM and consists of a "reset gate" and an "update gate" besides the input layer, but has no output layer. LSTMs performs faster than GRU but GRU are more computationally efficient than LSTMs. In figure 16, we can see how LSTMs and GRUs works.

![Figure 15: LSTM to the left and GRU to the right, source: [48]](image)

Another approach to use LSTM and GRU is to apply a Bidirectional "wrapper" [Keras web page] to create either BiLSTM [49] and BiGRU, so called "Bidirectional RNNs" [50]. BRNN behave differently from traditional RNNs, since it starts from both the beginning of the sentence, and the end of the sentence, to predict both the next word as well as the previous word in the sentence. However, this wrapper will double the computation time for our data to be trained by the BRNNs.
2.7 Evaluation Metrics

After a prediction has been made, in order to analyze how the model performed, one can use evaluation metric to receive information regarding the performance of the model(s). According to [35], there are 2 types of evaluation metrics that can be used:

1. **Confusion Matrix:** This is defined as "a table that is often used to describe the performance of a classification model on a set of test data, for which the true values are known" [35]. It will present how each class has been predicted in comparison to the true classes; This is best applicable for multi-class problems. For binary class problems, there are 4 performance metrics: true positive and negative (TP and TN) and false positive and negative (FP and FN).

![Confusion Matrix](image)

From these metrics, we can generate 4 types of scores that can explain the result of the predictions: Accuracy, Precision, Recall and F1. An alternative to a confusion matrix would be a
classification report that contains all the prediction metrics, but presents the results based on percentage.

**Accuracy** is the sum of true positive and negative divided by both true and false positive and negative:

\[
\frac{TP + TN}{TP + TN + FP + FN}
\]

The downside of accuracy is that it can suffer from class-imbalance problem, where one class of data is smaller than another one [52]. In accordance with [35], the metrics precision and recall are considered better for this kind of problem.

**Precision** presents the amount of correct prediction:

\[
\frac{TP}{TP + FP}
\]

where the value generated by the equation has to be as high as possible.

**Recall**, also known as sensitivity and True Positive Rate (TPR), it generates a result of how much the model has predicted correctly; the equation is

\[
\frac{TP}{TP + FN}
\]

and has to generate a value as high as possible; Also worth mentioning that there is also a False Positive Rate (FPR) that can be useful later where it calculates:

\[
\frac{FP}{FP + TN}
\]

**F1 score** is used to compare two classification methods with each other using this formula:

\[
F1 = 2 \times \frac{precision \times recall}{precision + recall}
\]

The values generated from this equation will range from 0 to 1 and the highest value determines the best classifier.

2. **ROC and AUC**: Receiver Operator Curve (ROC) and Area Under the Curve (AUC) are two very important metrics for evaluating the prediction from the models. ROC displays the TPR and how it compares with FPR in a graph by a line. The AUC of the ROC should
be close to 1, meaning that the ROC must not touch the linear line in order to be considered as a good prediction. ROC "shows the true positive and false positive rate for every probability threshold of a binary classifier" [35]. For example, by looking at figure 19, at the plot farthest to the right, we can see how the ROC curves are drawn with an AUC that is created by the line, which describes a 0% AUC, i.e. a problematic result. If the AUC is at 50%, the ROC curve will draw a linear line, which describes that the TPR and FPR are of equal value (0.5 or 50%). If the AUC is at 100%, the TPR is at 1 and FPR is at 0, which is a goal when trying to evaluate predictions.

Figure 18: Example of roc curves, source: [53]

2.8 Implementation using Data Science Process

The data science process consists of 6 steps that must be followed in order to perform implementations and those steps are according to [54]:

1. **Define the problem**, describe the situation and its issues and then brainstorm possible methods to be used as possible solutions. The methods can be inspired by other developers and modify them to suit your needs.

2. **Structure**, the gathered and unstructured data must be structured in a row and column based format with a finite number of rows but with a finite number of columns consisting
of a case number \[43\] the text and other data that can be used as outputs.

3. **Clean and Explore the data**, in the case of "clean", preprocessing of the data is a possibility. "Explore the data" is to, understand how the data is structured and what methods to apply for each output data.

4. **Model the data.** This is where we can apply classification methods to take the text data which has been "cleaned" from step 3, and apply text encoding methods such as (Bag of Words and TF*IDF) before we can insert the data into our model and then predict outputs based on the output data.

5. **Evaluate the model.** This is the process, in which evaluation metrics are performed. In the case of the deep learning models, we have to optimize the NN by performing hyperparameter optimization\[44\].

6. **Answer the problem**, this is where we conclude if the model predicted the desired result or not and if the model needs to be tuned or not.

---

\[43\] similar to private keys in Database tables

\[44\] See appendix D
2.9 Requirements Engineering

We will in this section present what tools are needed both in regards to hardware and software in order to develop our solution.

2.9.1 Hardware Elicitation

The hardware chosen for this work varied in terms of where the development and testing took place. The hardware that was used outside the hospital was a Mac Mini of 2018 with 8 GB of ram and Intel Core i5 CPU, and was used to develop the functions using a alternated form of text data. Using an alternated form of text data, satisfied the ethical aspects of this work since it didn’t contain any form of information that could connect to a physical entity. The hardware that was used at the hospital was a Windows computer, with an Intel Core i7 CPU and 16 GB of RAM. This computer was used to test the developed functions, and perform predictions based on given training data. By performing tests at the hospital, the results generated by the system, will be stored at the hospital.

2.10 Software Elicitation

We will describe in this section, the software APIs we have selected to use in our implementation as well as

2.10.1 Software APIs

This section will split up the requirements into APIs that has been used for implementing the solution. We will be using Python as our main programming language since it has a wide spread popularity, and it is a more compatible programming language for creating software that allocates hardware resources. Furthermore, there is a wide range of machine learning APIs that support Python. R is also a popular programming language but it doesn’t allocate hardware resources in comparison to Python.45

1. The data is stored as a table, in a CSV file format, which we can use Pandas for loading the data and segment it into multiple arrays called dataframes. Dataframes can be useful

45For more programming language comparisons see Appendix C section 4
since it contains methods of concatenation and can present multiple dataframes in one
frame, which can be beneficial in the long term.

2. We need to be able to preprocess our data. There are many steps in preprocessing: token-
enization, segmentation, lower casing and sometimes also removing stop words and stem-
ming. So, we have 2 APIs to test: regex or re and NLTK. We will do a comparison and see which one is more beneficial or efficient they are.

3. Classification is of great importance in this work and therefore Scikit-learn or sklearn is very useful. The reason for choosing sklearn is that it contains all the classification methods and it enables possibilities for evaluation of the model’s performance and metrics.

4. For predicting using deep learning, we will use text encoding software as well as Keras which enables creating ANNs for Deep Learning. Keras contains methods for using CNN, LSTM and GRU.

5. We will present some results using Matplotlib which plots diagrams using data and labels for convenience. SKlearn also contains the possibility for plotting confusion matrix as well as ROC AUC graph to apply our evaluation metrics.

6. To simplify the optimization of hyperparameters in the implementation of deep learning, we will apply Talos as a Search algorithm to help us find based on the results from the training.

2.10.2 Software architecture

The process is sequential, meaning that we have to implement the methods in a specific order so that it can be integrated later.

The process will be similar to the data science process step by step:

1. Extract text data

2. Apply preprocessing for the text

3. Apply feature extraction (Bag of words or TF*IDF) and feature engineering (LSI) and tokenization + padding for deep learning to transform text into numerical values

4. Create Classification/Neural Network models
5. Load the transformed data into the models and split it into training and test data for both training and predicting, based on how the model has learned from the training data.

6. Evaluate the prediction using the evaluation metrics

We can divide the processes into these sprints:

1. Extract and preprocess
2. Encode the texts
3. Create the models, predict and evaluate
4. Perform software testing, improve the code and optimize the hyperparameters

2.10.3 Text encoding software

Creating word vectors using a word embedding method is a good approach, however, it can be time consuming in most cases which is not suitable for this work. In [50] the developer created a program that would simplify the approach for creating pre-trained word vectors for both Word2Vec and FastText. The decision came to choosing fastText, since the text that has been written is more note based, which means the words written may or maybe not exist in the dictionary. It means we have to analyse the sequence of characters using character embeddings. The training of the words using FastText focuses on n-grams rather than the entire word generates a better result.

2.10.4 Security Aspects

The code has been stored during the development in GitHub, in a private repository, where we can store our code in multiple versions. One advantage with using Github is that the supervisor can download new commitments to our solution and testing it even if i am not present at the hospital for testing.

The development has been conducted in both Visual Studio Code and in 2 Jupyter Notebook files: one for the classification methods and one for deep learning methods; the reason for this is when i upload the code to GitHub, my supervisor has to pull these files from there and run

---

46 based on the mentioned theories from the earlier sections
them. He then has to push the results back to GitHub, where i will be able to record the results from the predictions. By doing this, we ensure that the final data is solely aggregated, and the evaluation results is just a collective result of a prediction without any information that can be connected to a physical entity. We will also include another Jupyter notebook file, that we will call the functions defined in these files. The reason for doing this is to make sure that our functions are independent and has to work on the data that is given by other parts of the system.

2.11 Software development

This work involves in developing a software solution that can be used in a CDSS. Therefore we need to define the principals of software development such as according to \[57\]: "a set of computer science activities dedicated to the process of creating, designing, deploying and supporting software". We will only focus on creating and designing, since it is more relevant for this work.

2.11.1 Software development process

I will be applying an Agile software development method with focus on Scrum where I apply weekly sprints in order to develop an appropriate solution. The reason of selecting Scrum rather than Kan-ban is that it had more convenience for this work; since i am having weekly briefings with my reviewer, he is the equivalent of a scrum master for this work. More importantly, it gives a possibility to reflect on what i have been doing and what can be improved for developing my solution.

Apart from developing, I will also be testing my solution using the software testing\[47\]. The reason why this is important, is to validate that we deliver a solution that satisfies our requirements, thus making sure it generates the right functionality. The testing process is also included in the scrum process because, if there is a function, a method or an API that doesn’t work, we need to invoke it into our next sprint to make sure the solution works.

\[47\]See software testing process in this chapter
2.11.2 Software Testing Process

The software testing process consists of 4 steps of testing: Unit testing, Integration testing, System Testing and Acceptance testing. Unit testing describes every function in a software solution as a unit. It is here that we test every unit by inserting data to verify that unit generates the data that satisfies the requirements of that specific unit.

Integration testing is where we make our units work with each other. Since we have multiple units, we must ensure that they work with each other; if one unit has generated a result and then transferring it to the next unit, that generates a result, then these units have become "integrated" with each other.

The third process of software testing is system testing where we test all the integrated function so that we start from one function and finishes in another. The purpose of the name "system" in system testing is that we have a fully integrated solution that takes an input is being parsed between functions, to generate a desired result.

Finally we will apply an acceptance test where we will analyse if the generated output satisfies our initial requirements, in this case the model is trained with a sufficient number of epochs so that it is neither over nor undertrained, or that our ROC curve generates curves with no separability.

2.12 Configuration and Implementation process

The configuration consists of installing the APIs using a command prompt. For installing the APIs, type: `pip/pip3 install API`. To enable Jupyter Notebook there are 2 ways: install from terminal or installing Anaconda. To use the virtual environment, we write an input to

---

48 The process mentioned is in the order described
49 We have two NLP approaches to test so we will have two systems to test
50 see figure 19
51 This process will be used at the University Hospital since they have the actual data
52 In this case, the terminal was used since it was implemented on MacOS
53 The configuration can be simplified in MacOS by creating a bash script file (.sh files), and inside this script file, add all the installation commands; then inside the command prompt just execute the script file
54 If the installation didn’t succeed, try and add sudo before the command
55 which is a platform for data scientists that contains possibilities to configure the required virtual environment
the terminal: conda activate venv, and to shut it down, type: conda deactivate. To use an API (and also other python functions) in the Python script file, use the import method (for the python scripts, exclude the (.py) from the filename while importing the file). E.g. To use the Pandas API to load the data, write this code: import pandas as pd.

The scripts were implemented by segmenting the code into multiple callable functions to apply the possibility of performing unit testing. The multiple steps described in the data science process section have been separated into functions, but in some cases, these functions had to be fragmented to ensure simplicity and to be able to be accessed by other functions.

2.13 Data Acquisition

To load the file into the Python code, we use pandas API to load the data as a dataframe with the pandas.read_csv method. We can print out the contents of the dataframe and find that we have 42 columns where column 1 is the "primary key" and column 2 is the free text that we can use as an input for the models later. The rest of the columns can be used as output data but some of the columns have different kinds of data. Some of them are of binary class, some are categorical, and some are numerical. We decided to transform the numerical data into categorical data to simplify the distribution of values into different categories. The numerical data are LastContactDays which registers when the patient had contact with a doctor and Age, which we separated into 6 different age categories based on infants, kids, teenagers, adults and seniors. By splitting the output data into multiple classes of data it will be more efficient to predict later on.

2.14 Text Encoding

This section will split into two to ensure that text cleaning and text encoding are two different topics. As a quick reminder, before engaging into preprocessing, the text contains numbers and therefore a data type conversion is required, where we convert the entire text into strings to simplify the preprocessing.

---

56 *as pd* is a way to create an alias for the imported code, and the functions inside the alias, are executed as e.g. pd.read_csv()  
57 For full explanation and description of the functions implemented, see Appendix A
2.14.1 Pre-processing

This section will focus on the pre-processing of the text, as well as exclude stemming and lemmatization, since we want the sequence of the text to generate what the probability of the next word will be, if a given word is allocated, i.e. we want to preserve the context of the text without too much modifications; However, we implemented two preprocessing functions to test: one without stop words removal and one function with removal, where the removal function comes from the NLTK API.

The function `pre_processing1` takes both the dataframe and freetext data and performs tokenization using: `word = list(sentence.split())`. To ensure that the text only consists of words we apply: `regex.sub(r'\W+','",word.lower())` which removed periods, commas and numbers, and the `lower()` function performs lower casing of the word. In `pre_processing2` we perform the same regex function as above, but instead of using `split` we use: `nltk.WordPunctTokenizer()` to tokenize the text and add a list of stop words: `nltk.corpus.stopwords.words('swedish')` to parse through the tokenized text to remove any stop words from there.

2.14.2 Testing

Preprocessing was performed using both stop words removal and without. During testing when performing deep learning, the ANN didn’t predict good results due to words that have been removed that are important for that sequence\footnote{sentence} to predict the next word. Preserving the structure of the sentence is key and therefore, removing words in this type of analysis\footnote{Deep learning} means that we lose context of the sentence and that leads to the models performs bad predictions.

2.14.3 Encoding

The encoding methods can be implemented from Scikit-learn library by calling: `CountVectorizer()` for BoW, and `TfidfVectorizer()` for TF*IDF; Inside the functions, there are parameters that can be modified later, but after we call these functions, we insert our text using: `TfidfVectorizer.fit_transform(text).toarray()`, where we generate a ma-
trix which we transform to an array where we apply the same procedure for CountVectorizer function. After transforming the data into arrays, another method that can be used is TruncatedSVD() where SVD stands for Single Value Decomposition, which is the Latent Semantic Indexing (LSI) to improve the predictions.

All methods that have been used for word embeddings originates from Keras. To encode our text, 2 functions where used: Tokenizer() and Pad_Sequences(). Tokenizer converts individual words into an index by taking how many words there are in the text as an argument, i.e. the maximum number of words. A benefit of using the Tokenizer function is that it can calculate the number of unique words in the text and index it, which is useful for creating the weight values for the NN. Padding is used to "extend" the length of sentences of smaller size into similar size of most frequent length by including zeros into the indexed sentence. The benefit here is that the zero values will not activate all neurons in the NN. After performing encoding on the input data, the output data has to also be encoded using the np.array() function. This is necessary, since the model wouldn’t predict without this function.

2.14.4 Testing

There are three variables that we need before inserting our data into the NN: number of words NB_WORDS, Maximum length of a sentence MAX_LEN and maximum length of characters MAX_SEQUENCE_LENGTH. We can get NB_WORDS by parsing through our dataframe using a for-loop where we use the Split(" ") function to calculate how many words exists in the data. Similarly with MAX_SEQUENCE_LENGTH where we use len(x) where x is our index in the for-loop for each individual character (including spaces). The MAX_LEN is calculated by parsing through our dataframe using a for-loop using len(x.split(’ ’)) where x is our index, then apply the max function to get the max length.

2.15 Explorative Data Analysis - EDA

There are two approaches to use EDA: Descriptive or Visual EDA; Descriptive is where we print the content of the data and present the result with text, whereas Visual, is where we

---

60 A testing was conducted, to compare the encodings with LSI and without. It turned out that the results improved using LSI and therefore it is essential since it is a method for feature engineering

61 We will discuss this in the result section
apply graphical visualization methods, in this case Matplotlib to plot graphs on how the data is organized. EDA can be applied on both the input data and the output data, but for convenience is to create functions for both input and output data.

2.15.1 Output Data exploration

The reason to explore the output data is to analyze the distribution of the classes. For a more convenient approach, the labels in each class must be balanced in order for our system to not have biases towards one label.

![Figure 20: output data example](image)

Note that figure 20 is just an example of how plots can be presented. The Matplotlib function takes arrays as an argument to plot a single graph, but can also cluster multiple graphs using the subplot argument. The output data in the dataframe are stored in arrays using a separate function, which we then can store in a variable, or "pickling" it using the pickle API. The newly stored arrays are then used to plot individual graphs as you can see above. In the plot above, we can see how "imbalanced" the data is i.e. not equally distributed between all data in the labels.

---

[62] categories, e.g. priority is one label
[63] We will put the code in Appendix A
2.16 Data Modeling

Data modeling is a term for creating a machine learning model to classify and train the text to generate outputs. This section is so dense that we’ve decided to split this work into two: one for the classifications (SKLearn) and one for the Neural Network (Keras), since we have used two different APIs to implement these methods. The classification works by "calling" (function call, e.g. LogisticRegression()) one of the classification methods, insert our input and output training data using the .fit(X_train, y_train) function and predict using our input test data and .predict(X_test). We looked up the LogisticRegression() function in the API documentation that contains default parameters and if you want to tweak any classifier functions, you have to just insert the parameters inside the functions.

2.16.1 Neural Network modelling

When developing the NN, there are three components that are required: input layer\(^{64}\), the neural network and the output layer\(^{65}\). To create the input layer, there are two ways: the first is to create an embedding layer using the length of the padded sentences, and the second one is to insert an embedding matrix. To create the embedding matrix we are going to use the text encoding software and the fastText corpus file, where it will act as an foundation, where it will cross reference with the encoded data for generating the weights\(^{66}\).

2.16.2 Implementation of embedding matrix

An embedding matrix is according to \(^{58}\) a "list of all words and their corresponding embeddings", i.e. after the encoding, the words will be organized in a matrix where the first row will represent the text for the first element from the CSV file and so on. The matrix will represent the entire text dataset and therefore won’t perform iterative work on the other text elements. The column will contain all the unique words found in the text and index every word every time that word is mentioned in that particular sentence. The dimension of this matrix will be 300 vertical\(^{67}\) multiplied by the number of unique words found in the text. The corpus file comes in .bin and .vec files which we have created customized function to map each word from the dataset.

---

64 or embedding layer
65 or Dense layer
66 to create an Embedding matrix
67 this is a default value
to the corpus using either: `load_vectors2()` for fasTtext or `load_vectors_word2vec()` for word2vec; There are two approaches in accessing corpus files: the first one is downloading and the second one is to use a method from [56] that generates words from Wikipedia where you can create how many words you want to index into your corpus. The method from [56] has more benefits than downloading since you can rescale the size of your corpus that you want, which has support for multiple languages so that you can train multiple embedding layers for multiple languages in the future.

The `load_vectors2()` function calls `load_vectors()` to load the fasTtext file and inside the function returns parameters such as: `word_to_vec_map`, which contains the values for each words to be stored in the embedding matrix. `words_to_index`, is where each word has an index value and can be swapped with the word2index variable generated from the Tokenizer function; `index_to_words`, is just an inverse of `words_to_index`; `vocab_size`, is the size of the unique words and can also be swapped with the MAX_LEN variable, and `dim` which will be used as the `output_dim` in Keras, and also the size of our embedding matrix. The next step is to create the embedding layer using the variables generated from the `load_vectors()` function by calling the `pretrained_embedding_layer` function. The implementation of the embedding layer will look like this:

```python
from Keras.layers import Embedding

First, we create an empty matrix:

```python
emb_matrix = np.zeros((vocab_size, dim))
```

Then we parse through each word in the `word_to_vec_map` to get the index to be inserted into the matrix.

```python
for word, index in word_to_index.items():
    emb_matrix[index, :] = word_to_vec_map[word]
```

After the parsing, the `Embedding` function is called, where it inserts "vocab" generated from the `Tokenizer` function, our `dim` variable, our MAX_LEN variable from parsing the dataframe, which we have to import `Embedding` to set the embedding layer

---

68 which we have to import `Embedding` to set the embedding layer
and we set trainable=False to freeze our weights so that it doesn’t change during training.

```python
embedding_layer = Embedding(input_dim = vocab_len, output_dim = dim, input_length = MAX_LEN, trainable=False)
embedding_layer.build(MAX_LEN)
embedding_layer.set_weights([emb_matrix])
```

An alternative to the embedding matrix is to define an Embedding() using the parameters acquired from the EDA_input() function: e.g. Embedding(134,128,input_length=134)

### 2.16.3 Implementation of a NN model

The Sequential() function is used to implement a layer in the NN since it enables methods to be added after each other. Here is an example of creating a basic LSTM model:

```python
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(vocab_len))
model.add(Dense(dense))
```

The Sequential function enables us to add functions such as LSTM() and Dense(), and in between the layers, it gives a possibility to add regularization methods, such as batchNormalization() and Dropout() to prevent the model from over/underfitting, but also add penalty functions such as kernel, bias or activity regularizers. Instead of LSTM, other NN models can be implemented, such as: GRU, Bidirectional(LSTM) or Bidirectional(GRU).

To complete the building of our model, the model.compile method is called, where a loss function, an optimizer and metrics are included to compute the same metrics as in the classification methods. This method will enable, according to back propagation that "alter the parameters of the neural network in the right direction". For the loss function,
binary_crossentropy was chosen where it is more suitable for the binary classification data. For the optimizer function, we chose to use a predefined optimizer from Keras called Adam, which contains these parameters: learning_rate=0.001, beta_1=0.9, beta_2=0.999 and no decay of learning rate. Finally, for the metrics function, we created customized metrics functions\(^{72}\) to be used to calculate the metrics and then will be plotted in the graphs since Keras only has metrics for accuracy. The input parameters for these customized metrics functions are the actual and predicted value of the data.

After the model has been compiled (i.e. created), the training data\(^{73}\) can be inserted into the model.fit() function, along with additional mandatory parameters: batch_size which takes samples of data and perform training\(^{74}\)\(^{75}\) epochs that trains the data a number of times and validation_data which is created by splitting the training data\(^{76}\). And finally, we call model.predict(), which takes on our test data and perform predictions.

### 2.17 Implementing a search algorithm

Since finding an optimal hyperparameter is time consuming, one alternative approach is to implement a search algorithm where it is enabling to type multiple parameters, and the search algorithm tells the model to train using only the declared parameters. One advantage using a search algorithm is that it creates multiple combination of models and presents only the result from the best performing model; There is also possibilities to extract and evaluate the best performing model based on that particular combination of hyperparameters. That is the reason why Talos is most suitable as a search algorithm for models created in Keras\(^{77}\).

\(^{72}\)for precision, recall and f1-score

\(^{73}\)Which has been split into training, test and validation data, where the training data must be larger than the test data since it is being used to train the model. The validation data is part of the training data, because it validates if the model has been trained properly

\(^{74}\)The default size of this parameter is 32

\(^{75}\)in the example data file, we had 13 texts as training data so the batch_size were set to 13, i.e. one batch

\(^{76}\)validation_split can also be used and give a value between 0 and 1 to distribute training and validation data

\(^{77}\)There is a search algorithm for SKLearn, but to use that algorithm, a conversion was required, which was time consuming and didn’t enable conversion back to Keras
2.17.1 Implementation of Talos

When implementing Talos, there focus was on one variable: params. Params will contain all the hyperparameters, which will be stored in arrays, where each parameter has a naming index and has its own values; e.g.

```python
params = {'activation': ['relu', 'elu'],
          'optimizer': ['Nadam', 'Adam'],
          'losses': ['logcosh'],
          'hidden_layers': [0, 1, 2],
          'batch_size': [20, 30, 40],
          'epochs': [10, 20]}
```

To perform a search this method is then decalred by using a dummy LSTM model as an example:
```
scan_object = talos.Scan(input_data, output_data, model=lstm_model,
params=p, clear_tf_session=False)
```

It will both create the model, import the parameters from the params variable and also the input/output variables. The variable scan_object will contain the result of the search algorithm where it contains the "best" parameters.

2.18 Evaluation metrics

We implemented one evaluation function for each approach (classification and deep learning) but the key aspects of these functions are to plot confusion matrices, classification reports and roc curves. We have discussed what a confusion matrix is, and how roc curves plots and classification reports describes the 3 evaluation metrics. The classification report in figure 23, covers both Bag of words and TFIDF, so for every method we use the title of the plots changes to display what methods we have used to predict. The confusion matrix in figure 24, will change its colors depending on the value (TN, FN, TP, FP) and will show the outcome of the predictions, inside the elements in the matrix. The ROC curves in figure 25, are examples of what the ROC curves will look like and in figure 25, we can see a bit of change which, if we compare it to figure 19, it will verify that our model is either not good or we need more data to train.

---

78 In terms of how small the loss value got from training
79 based on our dummy data
Figure 21: Classification report example using dummy data

Figure 22: Confusion matrix example using dummy data with true label on the y-axis and predicted labels on the x-axis
Figure 23: ROC Curve from testing with real data to display our intention of applying ROC curves
3 Result

In this episode, we will present the result of our work, based on our Data science approach. The first step is to define the problem, which is that we want to predict based on the text if a patient can be admitted to the hospital, and not being referred to a local healthcare center. The second step, is to gather the data in the CSV file, where we preprocess the text using text vectorization methods.

3.1 Text vectorization

We discussed the Keras preprocessing in an earlier section and we have mentioned the text encoding for both Bag of Words and TD*IDF. However, for both Bag of Words and TF*IDF there are 2 configurable parameters that were used to vectorize the data. The first one is analyzer, were we encoded the text based on either the word or character; the second one was: \( \text{min\_df} = 0.0, \text{max\_df} = 1.0, \text{ngram\_range}=(1,1) \) including, \text{sublinear\_tf} for TF*IDF, which replaces the calculation of TF with: \( 1+\log(\text{TF}) \). We can call the equation: the frequency approach; \( \text{min\_df} \) is the minimum document frequency value that our text that can be stored in the array; we set the value to zero which means that any word can be stored, \( \text{max\_df} \) set to 1.0 is the largest frequency of a text that can be stored, i.e. combined with the \( \text{min\_df} \) value, we can store any word in the text. \( \text{ngram\_range} \) set to (1,1) is the default value that stores unigrams in the array. Comparing the analyzer approach with the frequency approach, the frequency approach generated better ROC curves than the analyzer approach; The advantage of using the frequency approach is that you can customize the \( \text{max\_df} \) value to remove stop words (i.e. the most frequent words are the words that are used regularly).

We moved the architecture of the functions, along with the descriptions and specifications into Appendix A to reduce the context of this section, but we will name the functions here to explain in which order the functions were executed.

3.2 Exploration of Input data

We have over 21000 sentences to train and we used the \( \text{eda2()} \) function to explore our input data. The purpose of exploring the input data is to find a length of a sentence in the text based

---

\( ^{80} \)this is the cleaning approach from the method section

\( ^{81} \)i.e. single words
on the length of all the sentences in the text data. Finding the length in the text data also enables an understanding in defining the size of the input layer of our NN. Sentences with a length smaller than the frequent sentence will be applied padding after the tokenization process. To retrieve the maximum length of a sentence, a graph was created (figure 26) to present the frequency distribution of our sentences.\footnote{Frequency distribution is commonly used to find the words the occurs the most, but in this work, it has been tweaked to find a frequent length of a sentence}

The highest frequency indicates the largest sentence. In the x-axis, we have the length of the sentence and in the y-axis represents how many times that length has been measured in the sentences:

We can see here that the most frequent length of the sentences ranges between 100 and 150, and when we print the length out, the function returns 134, which will be the size of our input layer.

### 3.3 Classification model

Step 4 is to model the data, we have two different machine learning approaches: classification and deep learning which we will discuss in two part where we start with classification; We chose 6 models for classification: Logistic Regression, Naive Bayes Bernoulli, Gradient Boosting,
Random Forest, Decision Tree and Ensemble (that consists of: Logistic Regression and Naive Bayes Gaussian). The configuration of each model is described below:

- **GaussianNB()**: Standard configuration, no modifications;
- **LogisticRegression()**: Standard configuration, no modifications;
- **Ensemble/VotingClassifier()**: Standard configuration and a soft voting. The difference between hard and soft voting is that hard will generate a solid decision on which algorithms generated a majority output, while soft voting generates a probabilistic mean value from the classification models to be used as a output. During testing we can see that soft voting enabled the creation of the roc plots.
- **DecisionTreeClassifier()**: we used `random_state = 0` where 0 is the seed for the random number generator, it can be set to `None` and it can randomize the order of the data.
- **GradientBoostingClassifier()**: We used the standard configuration.
- **RandomForestClassifier()**: We used 50 estimators where each estimator is one tree branch and `random_state = 1`

### 3.3.1 Execution Process

We divide the execution process into a list of order, to preserve structure and consistency of the execution process. The functions we are naming here are mentioned with full specification in Appendix A.

- **Preprocessing and Encoding**: For the preprocessing, we are using `Preprocessing1()` and then execute `text_processing()`. However, we are executing the `text_processing()` twice: one for BoW and one for TF*IDF; we use then the encoded data and the output data to split into train and test data using `feature_engineering()`.

- **Model creation and prediction**: We select a classification model, e.g. NBG, which is the Naive Bayes Bernoulli model, and then we call first the `predictor()` function to split the call to `train_predict_model()` into 2 (one for BoW and one for TF*IDF); The result from that execution is returned back as a result array.

- **Model Evaluation and plots**: The result array is then inserted into the `generate_metrics()`
function using the same acronym (NBG) and the result array, to generate results such as the classification report and the confusion matrix in 2 ways (through printing out on the screen or visual graphical plots that we can save in files). We are then calling the `plot_roc()` function to plot the ROC curve for both BoW and TF*IDF in the same plot.

3.4 Deep Learning model

We chose 6 models for deep learning: LSTM, GRU, BiLSTM, 2 GRU layers, CNN with LSTM and CNN with BiLSTM and the parameters used to model our Neural Network are

- Output dimension: 100 and 128
- Optimizer: Adam
- loss function: Binary Cross Entropy
- metrics function: F1-score, Precision, Recall and Loss.
- Dropout: 0.1
- Activation function for the hidden layers: Tanh (Default for both LSTM and GRU) and ReLU
- Activation function for the output function: Sigmoid
- Batch size: 64
- Number of epochs: 3

3.4.1 Execution Process

We divide the execution process into a list of order, to preserve structure and consistency of the execution process. The functions we are naming here, are mentioned with full specification in Appendix A. We start with the deep learning approach by calling `eda2()` to analyze our input data and to retrieve some parameters to be used as global variables.\(^{83}\)

\(^{83}\)Originates from Appendix E

\(^{84}\)Note that we are performing everything mostly inside the `word_embeddings()` function
• **Preprocessing and Encoding:** First we are calling on the `feature_engineering()` to create train and test data for our input and output data. The splitted input data are inserted into `tokenizer()` to encode our text. We are enabling special character removal and lower casing as we did in `Preprocessing1()`. Along with the tokenized data, we are generating a word to index dictionary that can be used to create an embedding matrix. After the tokenization, we use the `padding()` function along with `MAX_LEN` which is our global variable for the maximum length of a sentence; We will take the padded text along with our `y_train` data to call `feature_engineering()` again to create validation data. To create the embedding matrix for our CNN models, we call the `load_vectors2()` function to generate a word to vector mapping. This mapping will be used to analyze our dictionary of tokenized words in the `pretrained_embedding_layer()` function. However since we are using Talos, we need to pickle the matrix and load the pickle file inside the model functions of `CNN1()`, and `CNN2()`, so that it doesn’t affect the function call of the scan function in talos.

• **Model creation and training:** We create our models depending on if we use Keras (in the `word_embeddings()`) or SKLearn (using the `train_predict_model()`). Inside the function, we are performing our talos scan method (along with the selected hyperparameters in the `params` variable) to find the best model. To do that, we call the function that correspond to our selected model and we plot the performance of the training using `plot_performance()` function. Then, after the model has been trained, we will return the selected model to be used to predict.

• **Model prediction evaluation:** We are taking all the training, test and validation data as well as the model from two different models, to compare them side by side by calling the `we_evaluation()`. First we run predictions, then we print out a comparison between the predicted output and the actual output. To complete our evaluation, we plot out the classification report, the confusion matrix as well as the ROC curve and store it in individual files.

---

85 Only for deep learning
86 e.g. if we want to train LSTM, we would call `lstm_model()`
3.5 Results from predictions

The fifth step is to evaluate our models. After the implementation we apply our models with the data at the hospital. We can then perform training and, based on the training, we can perform predictions and evaluate the results of these models. The results of the evaluation metrics is presented in the table below where we cover all the evaluation metrics presented in the earlier sections. I.e. the previous steps mentioned in this section are applied (text transformation, training and prediction) and presented in the table below. The result of the predictions will be indexed for each model to present which one performed the best. We will present some plots for these predictions in Appendix C.
## Result from predictions based on `hosp_admit` label

<table>
<thead>
<tr>
<th>Model/Metric</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1_Score</th>
<th>ROC_AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a. LogReg + BoW</td>
<td>0.663</td>
<td>0.640</td>
<td>0.605</td>
<td>0.622</td>
<td>0.711</td>
</tr>
<tr>
<td>1b. LogReg + Tf*IDF</td>
<td>0.680</td>
<td>0.669</td>
<td>0.596</td>
<td>0.630</td>
<td>0.741</td>
</tr>
<tr>
<td>2a. BernoulliNB+BoW</td>
<td>0.681</td>
<td>0.668</td>
<td>0.624</td>
<td>0.642</td>
<td>0.739</td>
</tr>
<tr>
<td>2b. BernoulliNB+Tf*IDF</td>
<td>0.681</td>
<td>0.661</td>
<td>0.624</td>
<td>0.642</td>
<td>0.739</td>
</tr>
<tr>
<td>3a. Ensemble+BoW</td>
<td>0.683</td>
<td>0.663</td>
<td>0.628</td>
<td>0.645</td>
<td>0.737</td>
</tr>
<tr>
<td>3b. Ensemble+Tf*IDF</td>
<td><strong>0.684</strong></td>
<td><strong>0.665</strong></td>
<td><strong>0.623</strong></td>
<td><strong>0.644</strong></td>
<td><strong>0.746</strong></td>
</tr>
<tr>
<td>4a. Decision Tree+BoW</td>
<td>0.587</td>
<td>0.550</td>
<td>0.539</td>
<td>0.544</td>
<td>0.583</td>
</tr>
<tr>
<td>4b. Decision Tree+Tf*IDF</td>
<td>0.579</td>
<td>0.542</td>
<td>0.532</td>
<td>0.536</td>
<td>0.576</td>
</tr>
<tr>
<td>5a. RF+BoW</td>
<td>0.668</td>
<td>0.677</td>
<td>0.528</td>
<td>0.593</td>
<td>0.718</td>
</tr>
<tr>
<td>5b. RF+Tf*IDF</td>
<td>0.662</td>
<td>0.676</td>
<td>0.502</td>
<td>0.576</td>
<td>0.717</td>
</tr>
<tr>
<td>6a. GB+BoW</td>
<td>0.649</td>
<td>0.685</td>
<td>0.432</td>
<td>0.530</td>
<td>0.703</td>
</tr>
<tr>
<td>6b. GB+Tf*IDF</td>
<td>0.648</td>
<td>0.685</td>
<td>0.430</td>
<td>0.528</td>
<td>0.701</td>
</tr>
<tr>
<td>7a. CNN+LSTMv1</td>
<td>0.638</td>
<td>0.623</td>
<td>0.510</td>
<td>0.561</td>
<td>0.677</td>
</tr>
<tr>
<td>7b. CNN+LSTM+emb_matrixv1</td>
<td>0.614</td>
<td>0.643</td>
<td>0.358</td>
<td>0.460</td>
<td>0.656</td>
</tr>
<tr>
<td>7c. CNN+LSTMv2</td>
<td>0.634</td>
<td>0.624</td>
<td>0.532</td>
<td>0.574</td>
<td>0.685</td>
</tr>
<tr>
<td>7d. CNN+LSTM+emb_matrixv2</td>
<td>0.601</td>
<td>0.573</td>
<td>0.465</td>
<td>0.514</td>
<td>0.627</td>
</tr>
<tr>
<td>8a. CNN+BiLSTMv1</td>
<td>0.634</td>
<td>0.594</td>
<td>0.598</td>
<td>0.576</td>
<td>0.673</td>
</tr>
<tr>
<td>8b. CNN+BiLSTM+emb_matrixv1</td>
<td>0.599</td>
<td>0.684</td>
<td>0.239</td>
<td>0.354</td>
<td>0.655</td>
</tr>
<tr>
<td>8c. CNN+BiLSTMv2</td>
<td>0.632</td>
<td>0.612</td>
<td>0.547</td>
<td>0.577</td>
<td>0.676</td>
</tr>
<tr>
<td>8d. CNN+BiLSTM+emb_matrixv2</td>
<td>0.609</td>
<td>0.642</td>
<td>0.339</td>
<td>0.444</td>
<td>0.639</td>
</tr>
<tr>
<td>9. LSTM</td>
<td><strong>0.648</strong></td>
<td><strong>0.627</strong></td>
<td><strong>0.550</strong></td>
<td><strong>0.586</strong></td>
<td><strong>0.698</strong></td>
</tr>
<tr>
<td>10. BiLSTM</td>
<td>0.642</td>
<td>0.614</td>
<td>0.590</td>
<td>0.592</td>
<td>0.687</td>
</tr>
<tr>
<td>11. GRU</td>
<td>0.642</td>
<td>0.596</td>
<td>0.673</td>
<td>0.632</td>
<td>0.689</td>
</tr>
<tr>
<td>12. BiGRU</td>
<td>0.639</td>
<td>0.616</td>
<td>0.569</td>
<td>0.592</td>
<td>0.688</td>
</tr>
</tbody>
</table>
4 Analysis

We will in this section, present what techniques have been used to generate the results that has been presented in the table, along with comparisons, and explanation of the ROC-AUC score. There are definitions that are written in the table that describes the following acronyms:

- **RF** = Random Forest
- **GB** = Gradient Boosting
- **NB** = Naive Bayes
- **Ensemble** = Logistic Regression + Naive Bayes Bernoulli using a soft voting classifier
- **LogReg** = Logistic Regression
- **v1** = MaxPooling
- **v2** = AveragePooling

We performed two different kinds of machine learning approaches (classification in 1-6 in the table, and deep learning 7-12 in the table). The best model is determined based on the highest ROC_AUC score in each learning approach. The ensemble+TF*IDF was the winner for the classification models since it combined 2 of the fastest models (Logistic Regression and Naive Bayes Bernoulli) combined with a soft voting to generate a descent prediction result and the vanilla LSTM was the winner for the deep learning predictions.

BoW gives a value based on how many times it occurs in the text, while TF*IDF counts the frequency of the word occurring, the lower frequency, the lesser the word occurs. In the table, some classifiers performed better with BoW, and others, performed better with TF*IDF. In the next section will present in detail what comparisons where made in order to decide the best performing model in terms of the ROC_AUC score.

---

87 The logistic regression had a L2 penalty function, also known as ridge regression. This penalty function is an optimization parameter, and it is described in appendix D

88 The numbers are rounded to give a three decimal place accuracy
4.1 Traditional Machine Learning approaches

The low ROC score is describing that the classifier is not doing good predictions, e.g. the Decision tree generated a score lower than 0.59. It can be explained based on the evaluation, that the decision tree model may have overfitted\textsuperscript{89} RF performed better than GB\textsuperscript{90} since it handled overfitting better than GB, and therefore gave a better ROC score. The idea of using Ensemble was to try and combine multiple classification models, which generated a great result. However, combining models that took longer time to generate a predictions is not a good idea.

4.2 Deep learning approaches

In order to analyze how hyperparameters works and how to optimize them, multiple trial and errors were performed in order to study the behaviour of the losses during the training process. The initial step was to try out multiple hyperparameters according to the documentation from Keras. The selected hyperparameters were chosen based on the analysis and comparisons of each hyperparameter\textsuperscript{91} and then inserted into Talos.

The optimizers that was used initially was Stochastic Gradient Descent (or SGD), where there were approaches to modify the learning rate and the decay rate, and the losses didn’t decrease. An alternative to SGD was the Adaptive optimizers (Adagrad, Adamax, Adam\textsuperscript{92}). The optimizer that performed the best based on both proof of concept and the test was the Adam optimizer.

The epochs were modified to find if the model were improved during the learning. The initial number were 10 epochs, but the epochs didn’t improve after the 5th epoch, and in some cases when testing with larger than 15 epochs, the losses increases. The conclusion was based on the epochs, was to not exceed 5 epochs during the training.

The interval of the dropout were around 0 and 1, and since there were a small amount of hidden nodes, the goal was to have a very small dropout. The initial dropout value was 0.5 and

\textsuperscript{89}The data is balanced meaning that the classifier couldn’t make biased branches
\textsuperscript{90}The depth of the tree aka model were defined to 1, for simplicity and equality sake
\textsuperscript{91}They are defined in Appendix D
\textsuperscript{92}See Appendix D for in depth coverage of these optimizers
then performed attempts to decrease the value. The losses decreased the smaller the dropout got, which was a good indication that the training generated progress; the final dropout value became 0.1.

The activation functions were many, since there were 4 types of activation functions to select from: Softmax, Tanh, ReLU and Sigmoid. Since LSTM and GRU had Tanh as a default activation function, it was tested along with ReLU in order to analyse the behaviour of these NNs. Softmax and Sigmoid were initially tested, since softmax was selected based on proof of concept, and Sigmoid were selected also based on proof of concept. The losses generated using softmax generated a fluctuation, i.e. a loss where it both decreased and increased; the sigmoid presented a loss that didn’t fluctuate and therefore was more promising in its results.

4.2.1 CNN and RNN comparisons

When using a CNN, along with a RNN, three types of test cases: with or without embedding matrix, RNN or BRNN\(^{93}\) and averagepooling or maxpooling.

By looking at the table, not using an embedding matrix improves the ROC score since it large in terms of size and that leads to a longer execution time, but also the pooling is using a 3x3 kernel to extract values, which is not desirable, since we want to focus on the row which is a 1xN kernel.

The averagepooling generated better ROC scores than maxpooling, since it extracted an average value rather than the maximal value from the kernel. The assumption was that the maxpooling would generate a better ROC score, but no. Both pooling methods has a similar time execution complexity, i.e. both take the similar execution time.

In terms of RNN and BRNN, ordinary LSTM performed better than the Bidirectional one due to the fact that the ordinary RNN didn’t go backwards in the data sequence in the hidden layers. By not going backwards in a hidden layer, it reduces execution time and processing power.

\(^{93}\)This can also work without CNN applications
This section will discuss what went well, what went wrong and how this project can be improved further in the future.

The architecture of the deep learning models was a recreation from the imdb models from [60]. We replicated the models (including all of the hyperparameters) to ensure that the model doesn’t have any issues. The size of the training data from IMDB was of similar size to our dataset, but the input dimension in our dataset was higher since we had a larger amount of unique words. The larger input dimension affected the results since it generated smaller metrics values than that from the imdb dataset.

Our first implementation of deep learning embeddings was to include an embedding matrix to our training, but after multiple tests we came to the conclusion that the embedding matrix was mostly suitable for CNN. If you imagine that a image is a 2 dimensional grid of values, then CNN is the best candidate to train an embedding matrix. For the RNN we used a regular Embedding function call with the number of unique words, an output dimension and the maximum length of a sentence, but we included the RNN architecture to our CNN models so that we can emulate a regular RNN and a bidirectional RNN.

The Bidirectional models (BiLSTM and BiGRU) generated better results in both accuracy, precision, recall and f1-score since we are analyzing the sentences both from the beginning and the end. In some forums (such as Stack Overflow), there are mentions that Bidirectional NNs are creating 2 NNs for predictions. We can confirm this since we experienced a doubling in the output dimensions (from 100/128 to 200/256) in both BiLSTM and BiGRU, and therefore the training becomes a larger (in terms of execution time and performance), and that had an affect in the result table.

We tried both MaxPooling and AveragePooling in our CNNs for comparison and found out that AveragePooling performed better than MaxPooling in our tests, since we are using words that mostly has a higher word index value and that can affect the result of the RNN training in the next layers. Just like in image analysis, maxpooling takes only the brightest pixels to train from each kernel, while averagepooling takes the mean value from each kernel to train i.e.
a smoothed value from each kernel.

We tried to use the softmax activation function, as well as the sigmoid function, for our output activation function. During our testing, the generated loss value from the softmax function increased in comparison to sigmoid, and therefore gave us the conclusion that softmax is not suitable for this model. After we implemented the classification models, we started to model the NN and found out that the sigmoidial activation function was mostly suited for the output layer. The fun fact here is that during the studies in the machine learning course, I encountered a theory that said that an output layer with a sigmoidial activation function is called as a classification model. There is an approach called sentiment classification which is similar to the sentiment analysis approach we have been implementing and this fact didn’t change anything of our work, but sentiment analysis is the generic term for NLU and it generated better search results (during the research) than sentiment classification. When we were using Talos we compared 2 hidden layer activation functions: ReLU and Tanh. We found out that ReLU was giving us a better result than Tanh which was the default activation function.

For the classification methods, we tried to perform grid search and random search, and came to the conclusion that random search was indeed faster than grid search, but we failed to implement random search in Talos so we kept using grid search. We decided not to include grid search in the final code since it generated a resource exhaust error. We also implemented the grid search and random search algorithms from SKLearn into our Keras models, but first we had to convert the model using the KerasClassifier. We tested then the search algorithms which generated a lot of errors that made us question the reason why would we convert our model to be used in another API, so we researched again and found an alternative called Talos. Implementing Talos was not simple in the beginning since this method was found only 2 days before the final day of implementation. When we performed the scan it didn’t regenerate the Keras model with the best parameters since it was defined in the Scan function to finish the "tf session", which erased the model after the scan. That parameter was disabled in the end. The SVD which is part of LSI is used as a dimensionality reduction method and in the earlier stages of testing, we saw how the encoding resulted with or without LSI. It clearly improved the performance of the encoding of the texts as well as in training where we got faster training time than without SVD.
In the previous sentence, we mentioned something about sessions which are created every time we create a model, it will perform resource allocation to store all the parameters of our model and every time train a new model, more resources will be allocated, i.e. Keras threw a resource exhaustion error. In the worst case the python framework would generate a resource exhaust error after 2 created models. To solve this, we had to, after we had evaluated each model, clear the session using `Keras.clear_session()` so that every resource allocated from the keras API would be erased to make hardware resources available for new predictions. You could also implement a garbage collector and the `del` method to remove unnecessary data from the simulation.

We decided to use the length of 134 in the deep learning solution, since it was the most frequent length of all the sentences. The downside here, is that the sentences that were larger than 134, were excluded, since they were too large. The length of the sentence will be the length of the input layer in the NNs, and a larger sentence means a larger input layer. This will in the end become very resource heavy and take longer time to execute since we have a larger input layer. One solution is that the dispatchers write a sentence with a finite number of words and pad out if the limit hasn’t been reached.

In the early stages of implementation, we implemented Support Vector Machine using the linear kernel (`LinearSVC` = a binary classification model with a linear discriminant), but after 2 hours of training, the Jupyter notebook crashed so we had to swap SVM with decision trees.

We performed 2 types of CNN predictions: one with regular embedding and one including the embedding matrix. It did perform very badly in our predictions since we were using fasttext data that originate from wikipedia and therefore it contains a collection of words that is mostly suitable for the general person. Our text contains words that are useful in a specific context and therefore some of the words couldn’t be included in the embedding matrix. Another thing to consider is that CNNs have different pooling methods that extract data and we decided to only test MaxPooling and AveragePooling since they are the fundamental pooling methods that have knowledge about, and there are other pooling methods in Keras to experiment with. In Keras there are pooling methods that can be used not only in 1D but also in 2D and 3D. The difference between 1D and 2D is the size of the kernel and pooling we want to extract from our matrix (e.g. matrix with size 20x300 and using MaxPooling)
During the testing of our models, we tried to emulate models that could generate similar results in order to ensure our model works without any issues. For me as the tester, I can use a large dataset that is not the hospital dataset. The reason for this is that, to validate the dataset, the tokenization, the padding, the training of the model and the training metrics so that it works properly. We implemented a model similar to [60] and used the IMDB dataset that is inside Keras to train and predict the sentiments. The result of the validation was similar to [60] where they used only GRU but we tested both LSTM and GRU models for the IMDB dataset. However, we developed our model further to analyze which hyperparameters were better suited for our data, since the IMDB data was already tokenized and then applied padding. But as our dataset has different lengths of sentences and unique words, we therefore needed to modify the parameters to satisfy our requirements.

We must also apply Cross Validation to our model. When building and training a model, one important step is to analyse if the model is stable i.e. "how well it would generalize to new data" [61] with the goal to not pick up noise from data, or in the case of the model doesn’t over and underfit. This process is called validation. In machine learning, there is a method called cross validation that assesses how the model "will generalize to an independent data set" [61] and it is used to estimate the accuracy of the models predictions when it comes to new data.

We apply K-fold methods when we train the data by creating a partition of our data whose sole purpose, is to be used for validating the training of the model. There are a couple of approaches to apply Cross Validation according to [61]: Train/Test split (Holdout), K-fold, leave one out and Stratified K-fold. Holdout has been used in this work where we split the data into train and test (and sometimes validation data) and has been implemented in code (see Appendix A). The downside of splitting the data for train and test (and validation) is that there is not enough data to train. K-fold takes the data and splits it into K different holdouts, where we take a portion of the data to be holdout, train the model and doing it K times where the final accuracy of the model is based on the average of the multiple predictions. After each iteration the partition that was used for training is changed, which will ensure that all the data are trained and validated, but it can be power consuming depending on the scale of the data, so a consideration of how much data we want to apply K-fold. Leave one out is according to [61] a "special case of Kfold" where the K is "equal to the number of samples" in the dataset where
we iterate through all the samples but leave one partition out of the iteration. It can be useful if we have a small quantity of data. Stratified K-fold is a version of K-fold that will try and split the data into partitions with equal amount of data in every partition that we can apply training and prediction to. According to [61], Stratified K-fold can be useful for either small datasets, unbalanced datasets and multiclass classification. We used Holdout in this work but if we want to train other partitions and not rely solely on one then K-fold is a good alternative. To get the best result where the data is equally distributed you can apply Stratified K-fold.

5.1 Future Enhancements

We implemented a system that takes on binary classification problems, and had also intentions on implementing the same solutions for multiclass problems and tried it, but there are problems with training each label.

Another future enhancement is to make real-time predictions by writing text inputs to the system so it will make predictions based on the cause. That would enable the nurses to adapt faster to the changes of the medical conditions of the patient. But this would also require that we have multiple text analysis systems, where each one is trained on different topics/labels of body parts, organs, diseases and injuries to generate sufficient treatments for patients.

Another thing to consider is how the models improve during training of new data. We need to have a method that can help us monitor how good the model is becoming every time we insert new data to the dataset. For example, if we train the model, and then some time in the future train the model again with new data, we have to have a method that could monitor if the model is improving (generates better predictions) or not, so that we can decide if we need to modify the models/hyperparameters or not. To improve our CNN network, we can create texts of our own and create a embedding matrix that can be used to be trained by the CNNs. By having a record of all the texts we can index the words by inserting the text using the algorithms from [56] and use them to train the models using fasttext.

An alternative to Keras is to use the Tensorflow API (Keras is a part of tensorflow and uses its own methods) where we could have integrated the Keras network with some methods from Tensorflow, but Tensorflow has so much customizations it could make this work dense. The
only requirement for implementing a machine learning model was to use python, and learning Tensorflow and all of its components and features could have taken a longer time to learn how to tweak the network. In fact it took us a longer time finding an alternative to the search algorithms until the very last days of implementation. When we found Talos and made it work. What we are saying is that Keras is an interesting API for developing a NN, but we have in this work evaluated which classification model is best suited to take text and make predictions with it, since we have a dataset of a large size, and that requires a lot of power to train the model.

5.2 Conclusion

In this work, we have covered how text can be mined both in the traditional approach (of analyzing the rule based methods), and the computerized approach (using machine learning based approach). The computerized approach doesn’t take into consideration that the rules are defined by humans and understands everything in numbers. The NLP is a great way for machines to take text and perform computerized actions implemented by the developer, so we chose to implement sentiment analysis that takes texts and predict an outcome. Sentiment analysis was implemented using machine learning methods such as classification and deep learning, where we selected to first encode our text before the training and finally evaluated these models using evaluation metrics, classification reports, confusion matrices and ROC_AUC curves. We have in this work explored how we can implement machine learning methods to predict texts from dispatchers using the classification approach and deep learning approach, where we tested Bag of Words, TF*IDF combined with LSI for classifications and tokenization plus padding for deep learning for text encoding. Then we predicted 12 different models that ended up with 24 different predictions to analyze if we can get better results by combining two different classification models, as well as comparing CNN, RNN and BRNN models. To find the best hyperparameters without guessing, we used a search algorithm to help us find the best model. By training all these models we are getting an understanding of how we can proceed to improve each model. Either change the size of our dataset to improve training or find the hyperparameters that will give you better metrics scores. My thoughts on the classification algorithms is that some of them were either not working due to a large scale of data (SVM) or the ROC plots were not smooth due to how the evaluation metrics are calculated\footnote{See the decision tree ROC curve in Appendix C}.
A Code architecture

Since we split the models into 2 parts (classification and deep learning) we split the implementation into 2 parts: part 1 and part 2, and we have implemented functions to make the implementation simpler to distinguish these parts apart (except from the evaluation methods since these methods are similar to generate the evaluation data for our result table). These functions that are being mentioned here doesn’t have an execution order, they are just mentioned here with a specification of what these functions takes as input and what output they generate.

- **eda2** (void function), (input: dataframe); explores the input data in terms of the maximum length of both sentences and chars, plots a table with the frequency of the lengths of each sentence.

- **Preprocessing1**, (input: freetext from dataframe and the dataframe itself, output: cleaned text); performs preprocessing in terms of splitting of words and lower casing as well as removing all special characters.

- **feature_engineering**, (input: input_data and output_data, output: an array consisting of splitting the data into training and test data for both input and output \(X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}\)); We are using from sklearn the train_test_split() method using a 70/30 split of the data (70% training data and 30% test data)

- **tokenizer**, (input: freetext data from dataframe, \(X_{\text{train}}\).values and \(X_{\text{test}}\).values, output: tokenized train and test data and a word index) we use the Tokenizer function from keras which take the input text from the dataframe to index the words and to apply these indexes into our train and test data.

- **padding** (input: the tokenized train and test data, output: train and test data with lengths equal to the maximal size of the sentences from eda2) padding adds additional zeros to the array of the tokenized texts so that in the NN not all input neurons are activated. We are using the pad_sequences() from Keras.

- **we_output_data_transform** (input: output data \((y_{\text{train}}, y_{\text{test}})\), output: transformed data); we need to transform our data into arrays so that our NN can be able to predict, plus we are importing \(y_{\text{train}}\).values, \(y_{\text{test}}\).values.

- **text_processing** (input: input data from preprocessing1(), output data, pro-
cessing method (BoW or TF-IDF), enabling SVD (boolean), **output:** encoded text using the processing method with/without SVD) Transforming our preprocessed text into either numbers or frequencies and if selected applying SVD/LSI to reduce the dimensionality of the encoded texts; we are using here the CountVectorizer(), TfIdfVectorizer() and TruncatedSVD() from SKlearn. We end this function by calling the feature_engineering() function to split the data.

- **initiate_predictions()** (input: the encoded data, the classification method and multiclass (boolean), output: test data, predicted data, accuracy and predicted data for roc plot); the encoded data is split into the train and test data to be inserted into the train_predict_model() function.

- **train_predict_model()** (input: name of the classifier, X_train, X_test, y_train, y_test, multiclass (boolean), output: the result of the prediction, accuracy, the result of the prediction to be used in the roc curve); we check here the acronym for what classification method to be used: NBG = Naive Bayes Bernoulli, RF = Random Forest, GB = Gradient Boosting, tree = Decision tree, logreg = Logistic Regression and Ensemble = LogReg and NBG. The train data (X_train, y_train) are used to train the model and then predict using the test data (X_test, y_test) and then returns the results of the accuracy of the model, the predictions from both model.predict(X_test) and model.predict_proba(X_test)[:,:1] to plot the result to the roc curve.

- **predictor()** (input: An array of the encoded data from the text_processing() function, the selected classification method and multiclass (boolean), output: an array of results from the predictions) This function is used as a handler to call the initiate_predictions() function to predict a selected encoded text using the selected classification method.

- **result_from_predictions()** (void function) (input: an array from the predictions and the name of the selected model); we are using the prediction array to extract the predicted output and the actual output to print a dataframe structured output to analyze the result of the predictions, we are also printing out a dataframe consisting of the selected evaluation metrics: Accuracy, Precision, Recall and F1-Score and storing this dataframe into a file.

- **generate_metrics()** (void function) (input: prediction array and the acronym of the selected classification model); This is the function we are calling right after we call the predictor() function. It will take the acronym and generate the full
name of the classification model to save the results of the predictions into a file. In this
function, we call on the `plot_classification_report()` which plots a colorful vi-
sual of the classification report (using `matplotlib`) for both BoW and TF*IDF (but
also for two deep learning models); we also call a function to plot a confusion matrix
called `plot_cm()`; all these plots are also saved to files to be presented. It also calls the
`results_from_predictions()` function.

- **plot_roc()** (void function) (**input**: the result from BoW and TF*IDF predictions, the
  name of the encodings and the name of the classification model); We are using the
  `roc_curve()` and `auc()` from SKlearn as well as `matplotlib`. The `roc_curve()`
  function takes the predictions from `model.predict_proba(X_test)[:,1]` as well as
  the actual output to generate both the false positive rate (FPR) and true positive rate
  (TPR); these parameters are then inserted into the `auc()` function to compute the AUC
  (Area under curve), then we save the plots in a individual file (the order of the plot is
  following: `plot.savefig`, `plot.show` and `plot.close`, because there is chance that
  if mix up the order, you will get an empty file).

- **load_vectors()** (**input**: fasttext bin file, **output**: an array that maps each word from
  the bin file, a dictionary variable that maps each word with a number, a dictionary variable
  that maps each number to words and the dimensionality of the text (300)); This function
  will take a bin file that has either been downloaded from fasttexts web page or created by
  `[56]` to create a word vector mapping that will create an embedding matrix.

- **load_vectors2()** (**input**: fasttext bin file, **output**: an array containing the variables
  from the `load_vectors()` function); This function calls `load_vectors()` to return an
  array.

- **pretrained_embedding_layer()** (**input**: the word vector mapping and the word to
  index dictionary from `tokenizer()`, **output**: embedding matrix); We are creating an
  embedding matrix with the size of the length of the word to index dictionary and the
  embedding dimension; the matrix itself is empty at the beginning but we fill it up with the
  index values for each word it finds (the words it doesn’t found it will generate a random
  number ranging from 0 to the size of the embedding dimension).

- **precision()** (**input**: the predicted output and the actual output, **output**: a precision
  value based on the input parameters)
• recall() (input: the predicted output and the actual output, output: a recall value based on the input parameters)

• f1_score1() (input: the predicted output and the actual output, output: a f1-score value based on the input parameters) this functions calls both the precision and recall function above and performs the mathematical calculation mentioned in the report to return the result.

• date_time() (input: an integer (ranging from 1 to 4), output: a timestamp value); we are generating a time log so that we can analyze the deep learning models how long each model took from start to finish.

• plot_performance() (void function) (input: the name of the ANN model and the results from the training (it is called history by most developers)); We will take the results and plot multiple subplots based on the metrics defined in the models (loss, precision, recall and f1-score1 (the metrics functions above)), we save the files to be evaluated when the training is over so that we can evaluate what went right and what went wrong.

• lstm_model() (input: the train and validation data (X_train, X_val, y_train and y_val) and the params to be used in the Talos scan function, output: a trained model); We are creating here a basic LSTM model with the embedding parameters: Embedding(voclen, output_dim=params[‘output_dim’], input_length=MAX_LEN) where voclen is the number of unique words found in the dataframe, the output_dim is an output dimension that we use Talos to find and the input_length=MAX_LEN is the maximum length of words in a sentence that we found from eda2(). The LSTM layer contains an output dimension which we also test using the same output_dim=params[‘output_dim’] in the embedding layer and an activation function which by default is a hyperbolic tangent function (tanh) but we also test the ReLU function.

Since we have a larger voclen, we are using dropout and as the output layer, we add a Dense layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function.
After the model is created, we call the `model.compile()` function to enable backpropagation, include `binary_crossentropy` and the selected metrics to generate the results; note that in Keras there is only accuracy included as a default metrics function, to include precision, recall and f1-score, we included custom metrics functions (`precision`, `recall` and `f1_score` mentioned above). Then we print a summary of the model and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using `plot_performance()` and returns the model back to the `predict_model()` function.

- **gru_model()** (input: the train and validation data (X_train, X_val, y_train and y_val) and the params to be used in the talos scan function, output: a trained model); We are creating here a basic GRU model with the embedding parameters: `Embedding(voclen, output_dim=params[‘output_dim’], input_length=MAX_LEN)` where `voclen` is the number of unique words found in the dataframe, the `output_dim` is an output dimension that we use Talos to find and the `input_length=MAX_LEN` is the maximum length of words in a sentence that we found from `eda2()`. The GRU layer contains an output dimension which we also test using the same `output_dim=params[‘output_dim’]` in the embedding layer and an activation function which by default is a hyperbolic tangent function (tanh) but we also test the ReLU function.

Since we have a larger `voclen`, we are using dropout and as the output layer, we add a `Dense` layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function.

After the model is created, we call the `model.compile()` function to enable backpropagation, include `binary_crossentropy` and the selected metrics to generate the results; note that in Keras there is only accuracy included as a default metrics function, to include precision, recall and f1-score, we included custom metrics functions (`precision`, `recall` and `f1_score` mentioned above). Then we print a summary of the model.
and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using `plot_performance()` and returns the model back to the `predict_model()` function.

- **bilstm()** *(input: the train and validation data (X_train, X_val, y_train and y_val) and the params to be used in the Talos scan function, output: a trained model);* We are creating a bidirectional LSTM model with the embedding parameters: `Embedding(voclen, output_dim=params['output_dim'], input_length=MAX_LEN)` where `voclen` is the number of unique words found in the dataframe, the `output_dim` is an output dimension that we use Talos to find and the `input_length=MAX_LEN` is the maximum length of words in a sentence that we found from `eda2()`. To create the bidirectional layer, we call the function: `bidirectional()` and insert an LSTM model inside (using the same parameters as defined for the vanilla LSTM model).

Since we have a larger `voclen`, we are using dropout and as the output layer, we add a `Dense` layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function.

After the model is created, we call the `model.compile()` function to enable back-propagation, include `binary_crossentropy` and the selected metrics to generate the results; note that in Keras there is only accuracy included as a default metrics function, to include precision, recall and f1-score, we included custom metrics functions (precision, recall and `f1_score1` mentioned above). Then we print a summary of the model and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using `plot_performance()` and returns the model back to the `predict_model()` function.

- **gru_model2()** *(input: the train and validation data (X_train, X_val, y_train and
y_val) and the params to be used in the Talos scan function, output: a trained model); We are creating a bidirectional GRU model with the embedding parameters: Embedding(voclen, output_dim=params['output_dim'], input_length=MAX_LEN) where voclen is the number of unique words found in the dataframe, the output_dim is an output dimension that we use Talos to find and the input_length=MAX_LEN is the maximum length of words in a sentence that we found from eda2(). To create the bidirectional layer, we call the function: bidirectional1() and insert an GRU model inside (using the same parameters as defined for the vanilla GRU model).

Since we have a larger voclen, we are using dropout and as the output layer, we add a Dense layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function.

After the model is created, we call the model.compile() function to enable backpropagation, include binary_crossentropy and the selected metrics to generate the results; note that in Keras there is only accuracy included as a defualt metrics function, to include precision, recall and f1-score, we included custom metrics functions (precision, recall and f1_score1 mentioned above). Then we print a summary of the model and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using plot_performance() and returns the model back to the predict_model() function.

• cnn1() (input: the train and validation data (X_train, X_val, y_train and y_val) and the params to be used in the Talos scan function, output: a trained model); In the embedding layer we have 2 approaches: to use the same Embedding layer configuration as above, or include an embedding matrix that will contain the weight values to enhance the training of the model (but we must also include the parameter: trainable=False). We must first implement a convolutional layer (called Conv1D) with parameters such as filters=64 that is the dimensionality of our output dimension, kernel_size =5
which is the length of the 1D convolutional sample (i.e. we will have 64 convolutional samples of size 5), padding='valid' since we are using padding and strides=1 which is the default value since we are generating a 1D CNN and the equation for calculating the output dimension for CNN is according to [62]:

$$output\_dim = 1 + \frac{(input\_dim - kernel\_size)}{strides}$$

If you want to use Conv2D, then use strides = 2

Since we have a larger voclen, we are using dropout and as the output layer, we add a Dense layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function. The next layer is Maxpooling1D that will take the convoluted data from the Conv1D layer and takes the maximum value from each of the kernels to the next layers. We are using a pool_size of 4 since we extract 4 max values from each kernel. After creating the convolutional layer and the pooling layer we are implementing the LSTM layer using the same parameters as in our LSTM model.

After the model is created, we call the model.compile() function to enable backpropagation, include binary_crossentropy and the selected metrics to generate the results; note that in Keras there is only accuracy included as a default metrics function, to include precision, recall and f1-score, we included custom metrics functions (precision, recall and f1_score1 mentioned above). Then we print a summary of the model and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using plot_performance() and returns the model back to the predict_model() function.

- **cnn2()** (input: the train and validation data (X_train, X_val, y_train and y_val) and the params to be used in the Talos scan function, output: a trained model); In the embedding layer we have 2 approaches: to use the same Embedding layer configuration as
above, or include an embedding matrix that will contain the weight values to enhance the training of the model (but we must also include the parameter: trainable=False). We must first implement a convolutional layer (called Conv1D) with parameters such as filters=64 that is the dimensionality of our output dimension, kernel_size =5 which is the length of the 1D convolutional sample (i.e. we will have 64 convolutional samples of size 5), padding='valid' since we are using padding and strides=1 which is the default value since we are generating a 1D CNN and the equation for calculating the output dimension for CNN is according to [62]:

\[
output\_dim = 1 + \frac{(input\_dim - kernel\_size)}{strides}
\]

If you want to use Conv2D, then use strides = 2

Since we have a larger voclen, we are using dropout and as the output layer, we add a Dense layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function. The next layer is Maxpooling1D that will take the convoluted data from the Conv1D layer and takes the maximum value from each of the kernels to the next layers. We are using a pool_size of 4 since we extract 4 max values from each kernel. After creating the convolutional layer and the pooling layer we are implementing the BiLSTM layer using the same parameters as in our BiLSTM model.

Since we have a larger voclen, we are using dropout and as the output layer, we add a Dense layer with the output value of 1 since we are performing predictions of binary values; if we would perform multiclass predictions we would have a value for each of the output values for that particular label (e.g. 5 labels = output value of 5) and a sigmoidal activation function.

After the model is created, we call the model.compile() function to enable back-propagation, include binary_crossentropy and the selected metrics to generate the results; note that in Keras there is only accuracy included as a default metrics function, to include precision, recall and f1-score, we included custom metrics functions (precision,
recall and f1_score mentioned above). Then we print a summary of the model and initiate the training using the training and validation data, a selected batch size and epoch and a callback function where it will generate plots to display the progression of the training of the model for each epoch. Finally we plot the performance of the model using plot_performance() and returns the model back to the predict_model() function.

- **predict_model()** *(input: the training and test data (X_train, X_test, y_train, y_test), validation train and test data (X_val, y_val) and the selected ANN model, output: a trained model); We are here calling one of the selected ANN models above (lstm_model(), gru_model(), biLSTM(), gru_model2(), cnn1(), cnn2()) using the search algorithm Talos and some selected hyperparameters from the array params to call the scan() function. The validation is a fragment from the training data that is trained after the training data has been trained; the validation data will behave a bit differently (i.e. not be on the same y-points as the training data because we have a smaller amount of data). After the scan we will load the model with the best hyperparameters back into Keras to be able to predict results.*

- **word_embeddings()** *(input: input data from the data frame, the output data and the selected ANN model, output: the encoded input data, the split input and output data and the trained model) Here we are applying the tokenizer and the padding() function to encode our text, then use we_data_output_transform() to transform the output data and then use feature_engineering() to split our data into train and test data and then apply the same function again to create validation data from the training data. Then we load in our fasttext bin file to call on load_vectors2() function to create the embedding matrix to be used in the CNN models. We are then calling the predict_model() function using our training and validation data as inputs and at the same time started a timer for which to measure how long does it take to create a model and train it using a selection of hyperparameters; after the model has been trained we are returning the model and then in the end return all the parameters to be used in the we_evaluation() function.*

- **we_evaluation()** *(void function) (input: the output data from the word_embeddings() function from 2 models); we are predicting using the 2 models created and then comparing it much like the generate_metrics() function where we compare the predicted out-


put with the actual output, printing out the metrics and saves it to an external file, plotting both the classification report and the confusion matrix using `plot_classification_report()` and `plot_cm()`. The only difference is that we are not calling the `plot_roc()` function but implementing the same code that was in that function inside this function. Since this function is the last function to use the keras models, we have to clear the keras sessions since it will allocate a lot of ram memory by calling: `Keras.clear_session()`.
B Plot results

Figure 25: ROC curves on the decision tree

Figure 26: ROC curves on Naïve Bayes Gaussian
Figure 27: ROC curves on Logistic Regression

Figure 28: ROC curves on Ensemble
<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.66</td>
<td>0.73</td>
<td>0.69</td>
<td>4.6e+03</td>
</tr>
<tr>
<td>0.63</td>
<td>0.55</td>
<td>0.59</td>
<td>3.9e+03</td>
</tr>
<tr>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>8.5e+03</td>
</tr>
</tbody>
</table>

**Classification Report lstm**

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69</td>
<td>0.62</td>
<td>0.65</td>
<td>4.6e+03</td>
</tr>
<tr>
<td>0.6</td>
<td>0.67</td>
<td>0.63</td>
<td>3.9e+03</td>
</tr>
<tr>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>8.5e+03</td>
</tr>
</tbody>
</table>

**Classification Report gru**

Figure 29: Classification report of LSTM and GRU, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s
Figure 30: Classification report of BiLSTM and BiGRU, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s.
Figure 31: Classification report of CNN1 and CNN2, the top row represents the predicted 0s, the middle rows represents the predicted 1s and the bottom row represents the average of the 0s and 1s.
Figure 32: Confusion matrix of LSTM and GRU

Figure 33: Classification report of BiLSTM and BiGRU
Figure 34: Classification report of CNN1 and CNN2
C Machine Learning

By definition from [63], machine learning is defined as: "Algorithms that parse data, learn from that data, and then apply what they have learned to make informed decisions". Machine Learning is comprised of multiple "disciplines" [64] with the intention to learn. The term "learning" originates from biological life where animals and insects make decisions based on the environment, how the environment changes and how life interact with each other. In terms of machines, "learning" means that the system adapts, based on its "structure" (Hardware), "program" (Software) [64] and the data (in terms of inputs to the application) so that the system can generate more accurate results if any of these components improve.

There are multiple reasons on why machine learning can be useful, but we have selected a couple of reasons from [64]:

1. We want to achieve a particular result and sometimes defining each task to receive that output is very difficult. A developer can possess certain data as input, generate an output, without the knowledge that the data is related to each other or not, and adjusting the learning model is easier so that learning by examples rather than defining each task individually.

2. The ML model can find relations and correlations in the amount of data, even if the amount of data is very large.

3. The model is modular, meaning that it can be built on other models to create a model that can generate more accurate results, or new and undiscovered results.

4. Even if the hardware or software changes, the model will still work without remodeling.

5. The model can improve in performance if the hardware or software is upgraded.

C.1 Wellsprings of ML

Machine learning originates from multiple theories and principles that combines both mathematical, biological, psychological and computational disciplines [64] such as:

- **Statistics**, Statistical learning is [65] is a "mathematical perspective on modeling data with a focus on data models" that use probability and probability distributions to calculate
e.g. possible outcomes and also stochastic simulations.

- **Brain models** consists mostly of neurons that connect together to create a neural network, where these models represent "non-linear elements with weighted inputs" [64].

- **Adaptive Control Theory** is used to track changes in the parameters while they change during the computation process (i.e. parsing through the neural networks).

- **Psychological Models** are according to [64] focused mostly on how humans act based on multiple factors during learning, which is called "performance" of the learning model.

- **AI** is an acronym for Artificial Intelligence that is based on mathematics (statistics) and focuses on predictions of actions and its probability of observed actions (e.g. Hidden Markov Model, HMM).

- **Evolutionary models** are models that 'evolve', meaning that the model improves its performance regularly [64] and examples that evolve during learning are called genetic algorithm and genetic programming.

### C.2 Types of learning in ML

In Machine learning, there are multiple types of learning which are very different, and for relevance of this work: Supervised Learning and Unsupervised Learning [66]. Supervised learning focuses on "using experience to gain expertise" [66]. i.e. to learn from someone who knows in this case, the machine can learn from experience of the training data to "predict" [66], i.e. generate data, based on data that has been allocated for training the model, to fill out the missing information for the data that is classified as test data. In unsupervised learning there is, according to [66], no "distinction" between training and test data. Instead this type of learning organizes data into a structure and can be extracted as a "summary" or a "compressed version" [66] of it. In conclusion supervised and unsupervised learning are each other’s opposites.

### C.3 Deep Learning

Deep Learning is an "aspect" [67] of AI "that is concerned with emulating the learning approach that human beings use to gain certain types of knowledge. At its simplest, deep learning can be thought of as a way to automate predictive analytics" [67], i.e. Deep learning is a method that enables us to emulate the human brain in terms of thinking and learning. Predictive analysis
is using both historical data and new data to "forecast activity" [68], i.e. generate an outcome using these data. The learning models mentioned in the machine learning section can become good at generating results, but one disadvantage is that they need "guidance" [63], i.e. if the result becomes inaccurate, the developer will have to modify the parameters by hand to improve the results. But with deep learning, the algorithms determine by themselves if the prediction is correct or not.

C.3.1 Artificial Neural Network

Since deep learning is designed to emulate the human brain, it will analyze data with a logical structure and probability theories "continuously" [63], using an Artificial Neural Network is approximately similar to the biological neural network that exists in the human brain. In figure 51, we can see how the ANN is structured with an input layer, an output layer and hidden layers, and that all are connected to each node by lines which are called "weights". In the case of deep learning, there are more than one hidden layers and figure 51 represents the minimal number of layers in a deep learning network [69].

The neurons in the network generate a "sequence of real-valued activations" [70]. The notion of activation is that neurons are activated, which means that some neurons are enabled, and others are not. The neurons through the weighted connections are calculated through a multiplication, followed by addition of the input neurons and their connected weights, then summarize all

![Figure 35: An example of how ANN is structured, source: [69]](image-url)
neurons that connects to a specific neuron. But for that neuron to activate, a value (which is called a bias) is added to the calculation of the activation of that neuron. Finally, if the result is above a specific threshold defined by the activation function means that the neuron has been activated to be used as an input to the next layer.

\[ Y = \sum (\text{weight} \times \text{input}) + \text{bias} \]

Figure 36: The neuron calculation, source: [71]

When modeling our dataset in our neural network, we need to evaluate our result, and to do that, we need a loss function. A loss function is a "method of evaluating how well your algorithm models your dataset". The value generated by the loss function must be as low as possible to make sure that the prediction becomes accurate. There are multiple types of Loss functions and mentions a couple of those such as: Likelihood loss, Mean-Squared Error (MSE) and log loss (also known as Cross Entropy).

When generating a value to activate a neuron in the next layer, the neural network also calculates the error that determines "how far is your model output from the actual output". When we calculate the values for e.g. the input layer to the first hidden layer, the values in the input layer and the first hidden layer are already defined, but the weights are not. The weights are assigned with random values and if the calculated value is not equal to the values to the next layered neuron, then an error will generate an error. To solve this issue, the neural network is able to perform to return to the previous layer and modify the weights to reduce the errors, until the error is smaller than the predefined threshold. This is called Backpropagation.

In conclusion, choosing Deep learning instead of classic NLP approaches tends to "improve the incompleteness and over-specification of a hand-crafted feature", meaning that humans don’t have to go into the system and update the rules or modify the learning parameters all the time. The neural network can create its own rules by learning from all the data that parses to generate the desired output.

---

95 we will explain it in a separate section
96 but not all the neurons reach above the specific threshold to be activated
97 based on calculating the difference between modeled and actual output and then squared called "squared error"
C.4 ML - Programming Languages

There is a variety of programming language options to choose from and in \cite{75}, they list 5 programming languages\cite{98} ranked by\cite{76}:

- **Python** - Rank 1: Python is according to\cite{75}, a "platform agnostic" programming language, i.e. it is "adaptable"\cite{76} to any operating system it is implemented on. It is a programming language mostly suited for desktop, web and hardware related programming, giving an explanation on why this programming language is popular among data scientist and back-end developers\cite{75}. Python is one of few programming languages that doesn’t require semicolon, or in some cases parenthesis. However, it has a focus on indentation to create a structured code. The downside is that the code can increase in size and complexity, which in turn may lead to reduction in performance.

- **Java** - Rank 2: Java is a programming language mostly focuses on web, desktop as well as mobile, and provides features such as "bytecodes", which enables the code to be run on virtual machines\cite{77} and also "sandboxing". These aforementioned features makes testing and monitoring the code much easier\cite{78}. Java is also according to\cite{75} more secure than C++, but the tradeoff of this ability is that it is slower in performance. Java doesn’t require intent but requires semicolon to complete a row of code.

- **C++** - Rank 4: C++ is a language specifically designed for mobile, desktop and its hardware. It can utilise memory and processing power to deliver "resource intensive applications"\cite{76}. It is one of the oldest programming languages, apart from C that is the predecessor of Java and C++, but doesn’t have the sandboxing approach. However, the code developed in C++ is more complex as you have to allocate memory which is a substep that is excluded in Java.

- **R** - Rank 5: R is a desktop specific programming language that has been used mostly for development of data science based solutions and according to\cite{75}, can apply classification methods, regression methods and decision trees and data visualization methods. It is also platform agnostic just as python, but according to\cite{75} it doesn’t have so much documentation regarding machine learning. Also R doesn’t utilize the hardware on the computer to develop faster software solution.

\footnote{also open source}
• **JavaScript** - Rank 6: A solely web based programming language. It can be used to implement Tensorflow for machine learning tasks, and is mostly targeted for web and front-end developers. However, the downside with JavaScript is that it is a web based programming language, i.e. it cannot utilize the computer hardware, nor can’t implement back-end solutions, which the other programming languages can do.

C.4.1 Machine Learning APIs

The programming languages mentioned in the previous section can be used to implement machine learning solutions, and in [79] describes 10 different machine learning APIs that can be used. These APIs have been ranked by [79] in this order:

1. **Tensorflow** - (Programming languages: Python)
2. **Keras** - (Programming languages: Python)
3. **Scikit-learn** - (Programming languages: Python)
4. **Pytorch** - (Programming languages: Python, C++)
5. **H2O** - (Programming languages: Python, R)
6. **Caffe** - (Programming languages: C++)
7. **MLlib** - (Programming languages: Python, Java, R)
8. **Mlr** - (Programming languages: R)
9. **Deeplearning4j** - (Programming languages: Java)
10. **MXNet** - (Programming languages: C++, Python, JavaScript, R)
D Optimization of data

D.1 Binary vs Multiclass data

When performing supervised NLP tasks, we need to understand what type of data that is going to be used as output data in order for our text to predict that output. Classification is one approach but more important we have to select methods that are able to perform a correct classification. There are two types of classification according to \cite{35} which are binary and multiclass classification. Binary classification focuses on output data that only have two types of output values which we can call binary classes. Multi class classification are classes that contains more than 2 classes. Some classification APIs uses by default binary classification, but can also be used for multiclass problems by some tweaking in the code \footnote{Some APIs have the methods mentioned in earlier sections are able to perform multiclass classification}. The only downside is that the implementation of some evaluation metrics \footnote{We will describe it in a later subsection} for multiclass classification has to also be modified.

![Binary class vs multiclass](source: \cite{35})

D.2 Over- and underfitting

When performing training of both classification and deep learning, the generated result may not be what you expect from a machine learning model. If you only implement an input layer, one or more hidden layer/s and one output layer, you will not get the results you desire from these models; We would call this underfitting since we have only implemented the layers. In the image below, we can see how over, under and appropriate-fitting models are presented; There
is 2 key terms when dealing with over/underfiting and that is mentioned in \[80\] called "Bias" and "Variance". Bias is according to \[81\] "the ability of your model function to approximate the data", and Variance is "the stability of your model in response to new training examples" \[81\]; So in figure 54, the underfitted plot (the left plot) has a "High Bias" \[80\] meaning that it has problems in covering the points and the overfitted plot (the right plot) has a "High Variance" \[80\] since it covers all the points but doesn't generalizes well since it also covers noise and/or outliers.

One goal with the training is to train the model with hyperparameters that will prevent the model from over/underfitting. In \[82\], they mention multiple examples of methods; In the case of underfitting, the only solution is to increase the dimensions or layers.

For overfitting, there is multiple approaches to prevent overfitting according to \[82\]:

- Modifying the layers
  1. Using an Ensemble of classifiers that "combines a lot of classifiers to stabilize the prediction" \[82\].
  2. Use Dropout, when we are training the NN we train the network with batches and in each training batch, the network will "drop different set of nodes, so we can think that we have one unique classifier for each subset of selected nodes. But at test time, it will use all the nodes to combine all their predictions. Therefore, it can be view as a version of ensemble."\[82\]. In other words, the dropout will disable some nodes in a particular layer depending on the fraction a developer inserts, e.g. 0.3 corresponds
to 30% of the nodes in a particular layer will be disabled. Finally, there is 2 types of dropouts in Keras: regular Dropout and SpatialDropoutX (X = 1D/2D)

- Regularization methods, using either L1 or L2 regularization functions as penalty functions that reduces the error during each epoch. In Keras, when applying either L1 or L2 regularizer, we insert a value into these functions called "lambda", depending on its size, can change the behavior of the NN. According to [83], if the lambda value is 0, then we will have a model that behaves similarly to a least square method and if the lambda is of a large size, then it will generate more weight values thus leading to underfitting of the model.

  1. L1 regularization is also called "lasso regression" [83] and includes a "squared magnitude" [83] to the loss function.

  2. L2 regularization is also called "ridge regression" and includes an "absolute value of magnitude" [83] to the loss function. [83]

- Dimensionality reduction methods:

  1. Early Stopping is to stop the training if the metrics data (loss, accuracy, precision, recall or f1-score) doesn’t change during each epoch

  2. Modify Batch size: change the number of data to be trained in each epoch

  3. Increase training data, if you have e.g. a dataset of 19 samples, try to find another dataset that has a higher number of samples to improve the training.

D.3 Hyperparameter optimization

One approach to improve the training of machine learning models and prevent over/underfitting, is to modify the parameters in the model in order to achieve optimal results and that is called "Hyperparameter optimization" [84]. There are two types of parameters: regular parameters which is "internal to the model" [84] e.g. the size of the layers, the weight values, number of support vectors in the SVM, coefficients in the linear/logistic regression or the size of the dimensions while hyperparameters are "external" parameters that doesn’t affect the data, but affects the behavior of the model since the developer can insert values so that the behavior can change to achieve the desirable result (a well fitted model).
Examples of hyperparameters are according to [85]:

1. Learning rate
2. Momentum
3. Batch Size
4. Epoch
5. Dropout
6. Optimizer
7. Network Weight Initialization
8. Activation function
9. and Number of neurons in the hidden layer

D.3.1 Hyperparameter tuning

The approach above has been solely based on examples from open source code (mostly from Kaggle) where they have just selected an optimizer, loss function, batch size and epochs to generate good performance in their model; but if we don’t know which parameter to change is to modify the hyperparameters. We need to first compare which of the search methods is the fastest, but we also need to load our sequential model from Keras to be able to be used in Talos. The deep learning models was using Talos to find the best model using a selection of hyperparameters. After the scan, Talos generated CSV files containing the models that were tested and the hyperparameters that were configured for that particular model. Talos can find the best model and load it back into Keras to perform predictions, but we had to open up the generated file and analyze which model was the best one. We decided that the model with the highest ROC score is the best model performed which is one approach on how we can extinguish the best performing model from the worst. We saw that the Ensemble + TF-IDF of Logistic Regression and BernoulliNB performed best, since it generated the highest ROC_AUC score.

For the traditional classification algorithms, multiple combinations of parameters are inserted into arrays and call the model.fit() method. We can also use the parameter n_jobs=-1
where jobs is associated with CPU cores, and if we don’t know how many cores we have in our computer and want to use all cores, we just use "-1". The SKlearn models have different parameters since they are structured and behaves differently so we implemented different parameters for different models.

To ensure our parameters for the deep learning models will be used, in [85], they used a dictionary for the parameters to be used by the search algorithm. After applying the `model.fit()` method, we can generate a variable called `model.best_params_` that will contain which parameters gave the best possible result during the training and the parameters can be then used to modify the model. We start first with the batch sizes and epochs and follow it up with optimizer, learning rate, weight initialization, momentum, activation function, dropout and number of neurons to be inserted into the neural network.

The process to use the search algorithms is the following:

1. Create the model
2. Create the parameters
3. Load both the model and the parameters into the selected search algorithm
4. Train the model using `model.fit()`
5. Extract `model.best_params_` from the model
6. Load that model with the best parameters and perform predictions
7. Evaluate the prediction using the evaluation metrics

D.3.2 Explanation of hyperparameters

In this section, we will present the hyperparameters and how they affect the testing of the deep learning models, including a brief theory behind these parameters to ensure that we modify them to generate a better result.

- **Learning rate**: Learning rate is, according to [86] the parameter that "controls how much we are adjusting the weights of our network with respect the loss gradient". We are updating the weights based on the actual weight value, minus the learning rate defined
by the developer multiplied with the generated loss value from the training. Since this parameter is defined by the user, it is important to select a value that is sufficient enough so that we can reach the smallest loss. The reason for selecting a smaller value, rather than a large value, is that we want the model to converge to the smallest loss possible.

Figure 39: The learning rate will find the lowest loss value in the curve also known as the local minima. In figure 55, we can see two learning rates of different sizes (small in the above plot and large in the below plot). We can see that a smaller learning rate will reach to the bottom more steadier than the larger learning rate. But according to the smallest learning rate is not always the best, they mentions that a small learning rate takes longer time to reach the local minima. Therefore the desired learning rate must be small enough but is able to converge fast enough.

Figure 40: Multiple learning rates in a loss function.

\[ \theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1) \]

If \( \alpha \) is too small, gradient descent can be slow.

If \( \alpha \) is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

\[^{102}\text{aka local minima}^{103}\text{the } \alpha \text{ in the equation is the learning rate}^{104}\text{local minima, minima = minimum}\]
During training of the loss function, we can see how the learning rate will behave (in figure 56), and therefore gives developers the decision to either decrease or increase the learning rate. For example: the green line has a high learning rate and has converged to a minima but the loss cannot decrease anymore because of this, and therefore we need to decrease the learning rate. In Stochastic Gradient Descent optimizer, there is a parameter that enables the learning rate to decrease the learning during each epoch, but some other optimizers doesn’t have learning decay and therefore it is up to the developers to decide if they want to have a learning rate that decreases or keeping it constant.

- **Batch Size**: Batch size is the "total number of training examples present in a single batch" [S7], and a batch is a fragment of the dataset you insert into the NN. The size of the batch you decide to insert will depend on the memory of the computer (A large batch size will generate a memory error during the compilation). For example, if you have 20000 datasets, it is better to fragment the dataset into batches of the size of 2000. But dividing the dataset into batches is not enough, we have to also include number of iterations and epochs.

- **Iterations**: Iterations is "the number of batches needed to complete one epoch" [S7], i.e. we can train all the batches of data in one epoch. For example: if we take the 2000 batches and perform one iteration 10 times, we can take the next 2000 batches in the next iteration and insert that into the NN. In keras, it is called: steps_per_epoch, which we will set in this case to 10 if we want to iterate 2000 batches 10 times.

- **Epoch**: An epoch is a value "when an entire dataset is passed forward and backward through the neural network only once" [S7], the reason for this is that according to [S7] the weight changes once in one epoch and the more epochs. The more the weights will change and the curve will transition from an underfit curve to to an overfit curve. The key here is to find the number of epochs that is greater than 1 but sufficient enough that the loss curve doesn’t show an overfit model.

- **Dropout**: We have mentioned Dropout in an earlier section, but a developer needs to consider the size of the nodes in the layer and how many they want to disable using Dropout. Disabling nodes in a layer generates efficiency in computing since we don’t calculate all the nodes in the layer if we use dropout, but we would probably lose some data on the way.
• **Optimizer**: An optimizer is, according to [88] a reduction of the name "Optimization algorithm", which will help us adjust the loss to be able to reach the local minimum. There are plenty of optimizers in Keras and we have mentioned a couple of those earlier but those that generated descent results will be mentioned here but according to [88]: Adam is considered to be the best and we will present how well it performed or we need to modify the parameters inside Adam.

• **Activation function**: this will, according to [89] take the input nodes, multiple the weights with the inputs, insert the product into the selected activation function and generate an output or to be used as the input for the next layer. An activation function will enable some neurons in the next layer to be enabled to take the input from the previous layer and continue the training. There are, according to [89] many activation functions, but the most "popular" ones are in order: Sigmoid, hyperbolic tangent (Tanh) or rectified linear unit (ReLU), if we don’t apply an activation function, the model will generate a linear activation function which "has limited power and does not performs good most of the times" [89] [105].

• **Number of neurons**: The size of the neurons in one layer can affect the computation, and therefore find a number that is small but trains all the data from the input layer and after that we can modify the hyperparameters however we want in the upcoming layers.

• **Additional Layers**: This has also affects computation since it will generate a longer training time if we apply more layers. If the layers have increased the learning rate would not affect the loss. We need to be careful when adding a layer into the NN.

### D.4 Activation functions

The activation function decides if the neuron in the next layer can activate/being enabled; according to [71], there are multiple activation functions: Step, linear, Sigmoid, Tanh and ReLU. Each machine learning model can use one different activation function in each layer, and according [71], there are some examples mentioned: Sigmoid can be used when applying a classifier and ReLU to achieve "faster training process and convergence".

In [89], they explain different activation functions: The first one is the linear activation func-

---

[105] We will compare the 3 activation functions and see how it behaves later
tion, but it is mentioned that the linear activation function doesn’t perform well "most of the times" [89] and it is suggested to use the other functions which are non-linear. The reason to use non-linear activation functions is of two reasons:

1. It has a degree greater than 1 since it can reach degrees greater than one

2. It is "differentiable" [89], meaning that we can enable backpropagation of the neural networks.

![Image of Sigmoid function](source: [89])

Figure 41: The Sigmoid function, source: [89]

The downside of the Sigmoid function is according to [89]:

- Vanishing gradient problem. When we perform backpropagation, we calculate a gradient based on the learning rate so that the loss will become smaller since the smaller the gradient the better the results generated. The vanishing gradient problem occurs when we cannot update the weights due to a small gradient [90] since we have already reached a local minima.

- The outputs from Sigmoid would not be close to 0 since the curve (in the figure above) does not go to 0, it goes to 1.

- Sigmoid can "saturate" and "kill gradients" [89].

- It converges really slowly, meaning that you will need more epochs than necessary to get the desired loss value.

The hyperbolic tangent function (Tanh) is an alternative to the Sigmoid function. According to [89] Tanh is more "zero-centered" which means the output ranges from -1 to 1, it doesn’t saturate or eliminates the gradients and converges faster than Sigmoid; But it still suffers from the vanishing gradient problem.
To solve the Vanishing gradient problem, we can use according to [89] the "Rectified Linear Unit" activation function. In [89], they mention that ReLU has become popular in the development of deep learning models. The outputs will remain 0 until the input is above a certain threshold. In the left plot of figure 59, the ReLU is using the threshold value of 0 and the equation of the function is according to [89]: $R(x) = \max(0,x)$ and if $x < 0$ then 0 and $x > 0$ then $x=x$ as you can see in the equation in the left plot. ReLU is both "simple and efficient" [89], and for that reason it is considered to be the popular one among the three activation functions. However, according to [89], it is advised to use ReLUs in the hidden layers since it generates values of 0 which will not activate the nodes in the next layer and can "die" [89]. To solve this, is to use a more advanced activation function from Keras called "LeakyReLU" [89] (the center plot in figure 59), where we multiply the input value with a $\alpha$-value defined by the user, which will make sure that the neurons doesn’t "die" during training. There is also according to [32], in the output layer which is called Softmax or a "normalized exponential function" [32] where it transforms values from the last hidden layer into values in the range $[0,1]$ (the sum of values generated by softmax is 1). The values that are generated represents probabilities that can be used to activate the specific output neuron.
D.5 Optimizer

The machine learning process is, according to [59] a "global optimization problem", where the parameters consists of weights and biases, and has to be adjusted so that the loss function is reduced to a minimum and that can be achieved using optimization algorithms. In [88], they mention a couple of them: Stochastic Gradient Descent (SGD), Adagrad, Adadelta and Adam. SGD updates the parameters "for each training example" [88], this means that the training data is split into batches and the updates occur in every epoch. However, since it performs parameter updates frequently, it can cause the loss function to "fluctuate to different intensities" [88], i.e. the results will oscillate in each update. This can be bad since the loss function will keep overshoot [108] and not reach the local minima (see figure 60).

To optimize SGD, is to first decide a small learning rate, and in [88] we can include parameters (values) such as Momentum and "Nesterov accelerated gradient". Momentum (ranging between 0 and 1) which is different from the hyperparameters the loss function is according to [59] something that calculates "how close a particular neural network is to the ideal weight during the training process" e.g. categorical_crossentropy and binary_crossentropy the loss line will go up instead of down.
0 and 1) is able to prevent the oscillation of the loss by multiplying to the update equation. The reduction of oscillations occur as the momentum value increases every time the loss value is decreased and as soon as the next update value is greater than the previous update value, the momentum will stop updating, this will reduce the oscillations and according to [88], the desired momentum value is "0.9 or similar". The Nesterov method \cite{109} is a method that was created by Yuri Nesterov, and the proposition of his method was to "first make a big jump based on our previous momentum then calculate the Gradient and then make an correction which results in a parameter update" [88]. That update "prevents us to go too fast and not miss the minima and makes it more responsive to changes" [88]. Their definition of "it" is the update value that will be able to be updated, ensuring that reaching the local minima will be possible.

Adagrad is a optimizer that enables the learning rate to "adapt based on the parameters" [88]. I.E. the learning rate is different for every epoch "based on the past gradients which were computed for that parameter" [88]. The pros of Adagrad is that we can define the learning rate once and the cons is that according to [88] the learning rate constantly decreases due to the update equation generates a positive value, that can be a problem since smaller learning rates tend to converge very slow and the training time will be longer than expected.

Adadelta solves the decaying learning rate by modifying the denominator of the update equation to calculate not all previous squared gradients, but only a fraction of those. However, those values will decay during each time step, and according to [88] we can add a momentum that only affects "the previous average and the current gradient". The difference between Adagrad and Adadelta, is that we are including a momentum in Adadelta and preventing the learning rate from constantly decaying, which makes Adadelta better than Adagrad.

The optimizer that is considered to "outperform" [88] the other optimizers is Adam, and stands for "Adaptive Moment Estimation" [88]. It works similarly to Adagrad and Adadelta, which enables the parameters to adapt but also calculates a decaying average of the previous momentum. In [88], they describe the modified update equation for Adam using 2 momentum values, which are called "mean" and "variance", these uses a $\beta$ value and in the final update equation and a $\epsilon$ value is used. The default values are defined in Keras:

\footnote{it is called Nesterov in Keras}
• $\beta_1 = 0.9$
• $\beta_2 = 0.999$
• $\epsilon = 10e^{-8}$

D.6 Search Algorithm

To simplify the optimization of hyperparameters, there are 2 methods that can simplify the optimization process of the hyperparameters according to [84]: Random search and Grid Search. Grid Search performs multiple predictions, based on the values mentioned as parameters stored as arrays called a grid, and takes multiple combinations of these parameters and inserts it into the model to perform predictions. Random search differs from grid search since we will "provide a statistical distribution for each hyperparameter from which values may be randomly sampled" [84]. In figure 61, we can see how the grid layout vs random layouts behaves differently where the grid is more organized apart from random.

![Grid Layout vs Random Layout](image)

Figure 45: Grid search vs random search, source: [84]
References


