Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2021

Pedro Salgado

Extração de informação de saúde através das redes sociais

Generating health evidence from social media

Universidade de Aveiro
2021

Departamento de Eletrónica,
Telecomunicações e Informática

Pedro Salgado

Extração de informação de saúde através das redes sociais

Generating health evidence from social media

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Luís Oliveira, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e da Doutora Alina Trifan, Investigadora do Instituto de Engenharia Electrónica e Informática de Aveiro da Universidade de Aveiro.

o júri / the jury

presidente / president

Rui Pedro Sanches de Castro Lopes

Professor Coordenador da Escola Superior de Tecnologia e Gestão, do Instituto Politécnico de Bragança

vogais / examiners committee

Augusto Marques Ferreira da Silva

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

José Luís Oliveira

Professor Catedrático Instituto de Engenharia Electrónica e Informática de Aveiro

agradecimentos / acknowledgements

First of all I would like to thank my family and friends for the close support. I also want to thank my supervisor for the precious experienced advises and my co-supervisor for the patience and continuous supervision through out this work. Cheers to eRisk for year after year setting new challenges in social monitoring.

Palavras Chave

redes sociais, informação de saúde, processamento de linguagem natural, aprendizagem automática.

Resumo

As redes sociais são um excelente recurso para conectar pessoas, criando assim uma comunidade paralela em que fluem informações acerca de eventos globais bem como sobre os seus utilizadores. Toda esta informação pode ser trabalhada com o intuito de monitorizar o bem estar da sua comunidade. De forma a encontrar evidência médica nas redes sociais, começámos por analisar e identificar posts de mães em risco de depressão pós-parto no Reddit. Participámos num concurso online, eRisk 2020, com o intuito de continuar a participação da equipa *BioInfo@UAVR*, em que prevemos utilizadores que estão em risco de se automutilarem através da análise das suas publicações no Reddit. Construímos um algoritmo com base em métodos de Processamento de Linguagem Natural capaz de pré-processar os dados de texto e vectorizá-los. Fazendo uso de características linguísticas baseadas na frequência de conjuntos de palavras, e outros modelos usados globalmente, capazes de representar documentos com vetores, como o Tf-Idf e o Doc2Vec. Os vetores e a sua respetiva classificação são depois disponibilizados a algoritmos de Aprendizagem Automática, para serem treinados e encontrar padrões entre eles. Utilizamos vários classificadores, de forma a encontrar o que se comporta melhor com os dados. Com base nos padrões que encontrou, os classificadores prevêm a classificação de utilizadores ainda por avaliar. De forma a tirar o máximo proveito do algoritmo, é desempenhada uma otimização em que as stop words são removidas e paralelizamos os algoritmos de vectorização de texto e o classificador. Incorporamos uma análise da importância dos atributos do modelo e a otimização dos híper parâmetros de forma a obter um resultado melhor. Os resultados são discutidos e apresentados em múltiplos plots, e incluem a comparação entre diferentes estratégias de optimização e observamos a relação entre os parâmetros e a sua performance. Concluimos que a escolha dos parâmetros é essencial para conseguir melhores resultados e que para os encontrar, existem estratégias mais eficientes que o habitual Grid Search, como o Random Search e a Bayesian Optimization. Comparamos também várias abordagens para formar uma classificação incremental que tem em conta a cronologia dos posts. Concluimos que é possível ter uma perceção cronológica de traços dos utilizadores do Reddit, nomeadamente avaliar o risco de automutilação, com um F1 Score de 0,73.

Keywords social media, health information, natural language processing, machine learning.

Abstract

Social media has been proven to be an excellent resource for connecting people and creating a parallel community. Turning it into a suitable source for extracting real world events information and information about its users as well. All of this information can be carefully re-arranged for social monitoring purposes and for the good of its community. For extracting health evidence in the social media, we started by analyzing and identifying postpartum depression in social media posts. We participated in an online challenge, eRisk 2020, continuing the previous participation of BioInfo@UAVR, predicting self-harm users based on their publications on Reddit. We built an algorithm based on methods of Natural Language Processing capable of pre-processing text data and vectorizing it. We make use of linguistic features based on the frequency of specific sets of words, and other models widely used that represent whole documents with vectors, such as Tf-Idf and Doc2Vec. The vectors and the correspondent label are then passed to a Machine Learning classifier in order to train it. Based on the patterns it found, the model predicts a classification for unlabeled users. We use multiple classifiers, to find the one that behaves the best with the data. With the goal of getting the most out of the model, an optimization step is performed in which we remove stop words and set the text vectorization algorithms and classifier to be ran in parallel. An analysis of the feature importance is integrated and a validation step is performed. The results are discussed and presented in various plots, and include a comparison between different tuning strategies and the relation between the parameters and the score. We conclude that the choice of parameters is essential for achieving a better score and for finding them, there are other strategies more efficient then the widely used Grid Search. Finally, we compare several approaches for building an incremental classification based on the post timeline of the users. And conclude that it is possible to have a chronological perception of certain traits of Reddit users, specifically evaluating the risk of self-harm with a F1 Score of 0.73.

Contents

List of Figures

List of Tables

Glossary

OLAP	Online Analytical Processing
NLP	Natural Language Processing
TF	Term-Frequency
IDF	Inverse Document Frequency
TF-IDF	Term Frequency–Inverse Document Frequency
BoW	Bag-of-Words
CBOW	Common Bag Of Words
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
DM	Distributed Memory
DBOW	Distributed Bag of Words
PPD	Postpartum depression
GPU	Graphics Processing Unit
DL	Deep Learning
ML	Machine Learning
XML	Extensible Markup Language
ANN	Artificial Neural Network
DNN	Deep Neural Network
W2V	Word2Vec
D2V	Doc2Vec
BoW	Bag of Words
PCA	Principal Component Analysis
UNSL	Universidad Nacional de San Luis
CLEF	Conference and Labs of the Evaluation Forum
API	Application Programming Interface
SVM	Support Vector Machine
SVC	Support Vector Classification
GloVe	Global Vectors for Word Representation
BERT	Bidirectional Encoder Representations from Transformers
NDCG	Normalized Discounted Cumulative Gain
LightGBM	Light Gradient Boosting Machine
RBF	Radial Basis Function
JSON	JavaScript object notation

CHAPTER 1

Introduction

This chapter overviews the motivation and objectives of this thesis. We introduce the scenario under which it was developed and outline its main contributions.

1.1 Motivation

Social media has been progressing to the point of being capable of replacing face to face communication. Millions of people are in touch through social media, where they can have conversations, expose their thoughts, share media, search the news, and even purchase items. Nowadays, there are over 3.5 billion users worldwide, and it is expected that this number will go up to 4.5 billion by 2025[1].

All this data gave birth to the new area, Social Media Analytics, where vast amounts of data flow that go through the social network are analyzed so that valuable information is extracted. Data in the form of text represents a sizeable portion of all this data, and there are research areas that benefit from social media text analysis, such as healthcare. The advances in Machine Learning (ML) for Natural Language Processing (NLP) made this analysis possible in conjunction with computer performance growth. Nowadays, there is the possibility of processing whole datasets of text in a matter of seconds and providing them to a machine learning algorithm capable of finding hidden patterns within the dataset and build predictions based on the knowledge obtained. The Figure 1.1 shows the evolution across the years of processor performance. However, processor performance is only part of the equation. With Deep Learning (DL), came the need to parallelize operations and the Graphics Processing Unit (GPU) is perfect for this job. More recently, cloud computing, relies on a distributed network of servers that work together to achieve a more efficient and scalable training of a DL model.

[1] https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/

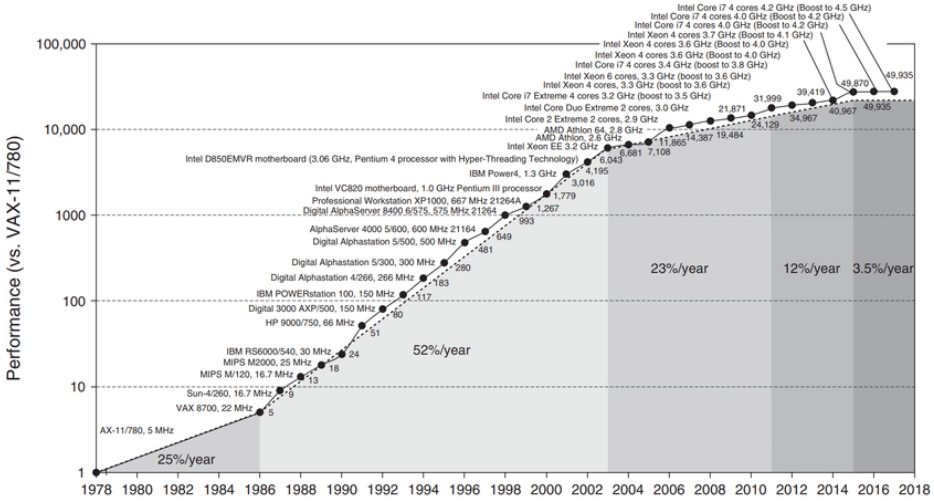

Figure 1.1: Growth in processor performance since the late 1970s. The chart plots the performance relative to the vax 11/780.[1]

Twitter[2] is usually a subject of use for analysis of social media because of its simplicity, public Application Programming Interface (API), and popularity. Even though it is only the 15th most used social network[3] over five hundred thousand tweets are posted every minute, originating the possibility of several studies. Ranging from extracting multidimensional data cubes for Online Analytical Processing (OLAP), as investigated by Mansmann, Rehman, Weiler, *et al.* in [2], to detecting health conditions such as flu, depression, pregnancy and eating disorders, as explored by Prieto, Matos, Alvarez, *et al.*[3].

Mental illness is a risk that everyone can be exposed to. World Health Organization reports that one in four people in the world will be affected by mental or neurological disorders at some point in their lives. There are several studies made that associate Social Media with mental health. One of the first studies that were made in this area analyzes how a higher use of the Internet can lead to social problems and even increase depression and loneliness. It is called the "Internet Paradox", [4] even though the Internet is a platform where communication is facilitated. This facilitation can also cause problems in social involvement and well-being. Perhaps with the ease of communication through the Internet, one could lose the motivation for face to face communication. Internet facilitated the emergence of Social Media, where millions of people expose their day-to-day lives. Facebook was founded in 2004 and since then has been evolving to the point of containing over 2500 million users worldwide, and this is just one of them. Twitter, Instagram, and Reddit are some other examples that also have billions of users. Since 1998, more studies came out to prove the point of the

[2]https://twitter.com/

[3]https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

article. For example, Pantic, Damjanovic, Todorovic, *et al.*[5] found a link between social networking and depression. More recently, in "The Social Dilemma"[4], people that worked in these big companies, Google, Facebook, and Twitter, claim that they are using manipulative strategies that have an impact on billions of people and are carefully designed to keep the user engaged and addicted.

The disrupt of one's well-being can lead to self-harm. In some cases, even to suicide, independently of being caused by social media or not. More than ever, we have the possibility of perceiving someone's mental state by only analyzing their online activity.

1.2 Conference and Labs of the Evaluation Forum

Conference and Labs of the Evaluation Forum (CLEF) contributes to the systematic evaluation of information access systems, primarily through experimentation on shared tasks[5]. Some of the challenges proposed by CLEF are CheckThat![6], in which the goal is to identify and verify claims on Social Media automatically. lifeCLEF and imageCLEF are other challenges, in which the goal is to automatically identify plants and classify images and annotate image data, respectively. Early Risk Prediction on the Internet[7] is the challenge that we will investigating and explores the evaluation method, effectiveness metrics, and practical applications (particularly those related to health and safety) of early risk detection on the Internet. In the previous three years, eRisk organized tasks that comprised processing sequentially texts written in Social Media and predicting different mental illnesses. This year, eRisk published two tasks.

- Task 1: Early Detection of Signs of Self-Harm
- Task 2: Measuring the severity of the signs of depression

The first task consists of sequentially processing evidence and detecting early traces of self-harm as soon as possible. Texts should be processed in the order they were created. In this way, systems that effectively perform this task could be applied to sequentially monitor user interactions in blogs, social networks, or other types of online media. In «A test collection for research on depression and language use», Losada and Crestani argues that tweets provide a little context about the tweet writer and that only a small portion of the writer's history of tweets can be retrieved. These authors choose Reddit[8] to create the textual collection for depression. The social network ranks 11th as the most used social network and usually is more of a forum, meaning posts should have more content to be explored.

[4]https://www.netflix.com/pt-en/title/81254224
[5]https://clef2020.clef-initiative.eu/index.php
[6]https://sites.google.com/view/clef2020-checkthat/
[7]http://early.irlab.org/
[8]www.reddit.com

1.3 Objectives

This thesis aims at studying and developing algorithms of social monitoring capable of extracting health-related outcomes. The practical scenario was the participation in a shared benchmark task for the early detection of online risk. In last year, BioInfo@UAVR took part in eRisk 2019, and the results are described in [7]. The dissertation focuses on improving these algorithms and targets a participation in eRisk 2020. The algorithm for the first task of this year's challenge, "Early Detection of Signs of Self-Harm" will be the main subject of study. This challenge is a binary classification problem, as there are only two labels. However, we should process texts in order so that we can emulate a better simulation of social networking. Metrics capable of valuing the speed of prediction are necessary so that the earlier a positive result yields, the better the score. In the next chapter, these will be presented. Before building the algorithm for Task 1, a preparatory project was built to get the background knowledge on the different explored areas necessary for the contest. This project also consists of a binary problem and aims at classifying postpartum depression. Both datasets are from Reddit. However, task 1 dataset is composed of all posts from users, and the dataset for Postpartum depression (PPD) is simply a collection of posts. Consequently, the prediction for Task 1 is more tricky, as some posts from a self-harm user might not be related at all to self-harm. The methodologies from last year will be studied and applied. Furthermore, new emerging practices in NLP and ML will be analyzed and compared.

1.4 Thesis Structure

This thesis, besides the current chapter, is composed of four more chapters:

- Background, chapter 2, describes the various topics necessary to have a basis on the matter, gives an overview of NLP, and enumerates the methodologies currently used. It also presents the big theme of machine learning to describe some of the most relevant algorithms and techniques.
- In chapter 3, Methods, we report the steps took to build two different projects on the matter of the thesis and present the results for each of them.
- Results, chapter 4, further offline experiments and analyses were made to achieve a better performance in terms of score and optimization, including optimization, tuning, and feature importance.
- Conclusion, chapter 5, discussion of the work done, suggestions for future work.

CHAPTER 2

Background

This chapter aims to give background knowledge on the topics relevant to this work. It overviews some of the methods currently being used both in NLP and ML. Essentially, to process natural language, we usually have three central units. The pre-processing unit is responsible for transforming text into more relevant data to be processed by an algorithm. After obtaining this more digestible form of text, usually called tokens, the next step is feature engineering. Valuable information can be extracted and derived from and then fed to our machine learning model in the form of vectors. This model must learn from these vectors and their respective labels so that later it can predict unseen vectors. These predictions are then used for evaluating the model's performance by comparing the predicted values to a golden truth. Of course, this is a very abstract and brief summary of the flows commonly used, and we will enter into detail in this chapter. The flow is exhibited in Figure 2.1.

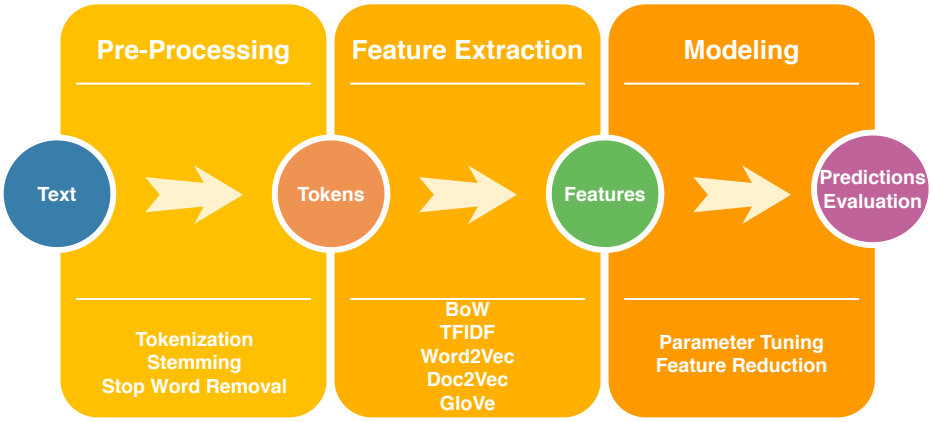

Figure 2.1: Common flow for processing and analyzing natural language processing.

2.1 Natural Language Processing

NLP is a subfield of artificial intelligence and is concerned with the processing and analyzing natural language. It started by having a set of hand-coded rules, which are still useful today if we are working with specific use cases. For example, suppose we were to do a sentiment analysis classifying each writing as positive or negative. In that case, we could set a rule to count the words "like" and "dislike" and obtain knowledge about the writing. Statistical NLP eventually came up and is currently the mainstream approach. It relies on large amounts of input data converted into a set of features and serves as input to what is usually covered by ML models. This strategy has the advantage of not being as specific and being adaptable and flexible. We can also use a combination of both strategies. For example, it is often the case that a rule-based system gathers the data for a supervised model. The Turing Test [8] is one of the first studies in NLP. In the paper, Turing describes a new way of testing a machine's intelligence, or at least the ability of pretending to be a human. Therefore human intelligence might be a more suitable term. Turing named it "The Imitation Game" and consists of three players: the machine, a human, and an interrogator whose job is to determine which of the players is the machine by asking them any question he wants. In the real world, the interrogators usually do not have such a wide field of possibilities as the imitation game, making differentiation much harder.

Chatbots are replacing humans more and more. These usually do not have such an abstract playground as the Turing test. The questions and answers typically focus solely on the entity the chatbot is representing. Take as an example, an airline company that wants to provide quick feedback to its clients for their repetitive questions, booking management, and other features. The machine could easily set a rule to understand a human trying to book a flight by analyzing the words that come after "from" and "to". This is a straightforward example, and the bot should be prepared to receive booking information in other words.

Nevertheless, we can perceive that this is achievable and is currently being worked on, for example by mindsay[1]. It should be noticed that when this happens, it is the machine doing the whole translation between human and machine language. What usually is done for booking a flight is, we go to the airline site and click on some buttons, enter locations of departure/arrival, choose the payment method. In this case, we are doing part of the translation. In the past and some cases of the present, a human agent is involved.

Now let us imagine the game from a different perspective, consisting of the machine, a doctor, and an interrogator who is also a doctor but pretends to be a patient. Let us

[1] https://www.mindsay.com/chatbot/airline

also suppose that this game's goal is to perceive the patient's disease and prescribe a treatment. Would the interrogator not distinguish between the players, then we could imagine switching the whole medical system that we know as of today. The machine already has a boosting start, which is the ability to process a patient's history in terms of diseases/genetics or even Social Media in seconds and compare them to other patients. eRisk is already proposing mental disease detection in the Social Media. One could imagine a feature in which a system, given the user's writing on Social Media, would detect with a certain confidence level diseases of the user in question. In NLP, there will be constraints because not all users write on Social Media, and even those who do might not expose their true selves through their posts. If we get out of the box that NLP is, various things can come to mind. A simple like, purchase, follow, a subscription can convey meaning to the classification. In this section, different ways of pre-processing text will be described, and various feature extraction techniques.

2.1.1 Tokenization

Tokenization is the process of segmenting text into words/tokens. There are several ways to do it. For example, splitting into white spaces would turn the string "He is writing" into "he", "is", "writing". Some sentences do not have a unique way of segmenting the words, should we preserve the hyphens and apostrophes? What happens to punctuation, and what about abbreviations and emojis? It makes sense to preserve hyphens and apostrophes since they can transpose into different meanings by removing them. Abbreviations and emojis can also convey meaning to a sentence, therefore in some cases, they should not be removed. Punctuation can have a role in understanding the emphasis of the phrase, tokens are often lower cased. The catch is that no tokenization method fits best in all algorithms, and to get the best results out of segmentation, several approaches should be attempted.

Stop Word removal refers to the process of removing words that are not needed for the analysis. In this way, we can save up space and time, which can be valuable for the project. However, these words can also remove meaning from the text. As an example, people with depression will tend to focus on themselves. Therefore, higher usage of "I", "me", "myself", usually considered stop words, might be present in a depressed person's text. This is valuable information and should not be removed if the system's goal is to predict depressive users.

A regular expression is a sequence of characters that define a search pattern that later can be passed to an algorithm capable of finding these patterns and, if needed replacing it. There is a built-in package for these operations in python, *re*[2]. It has multiple functions such as **findall()**, **search()**, **split()** and **sub()**. For tokenization

[2]https://docs.python.org/3/library/re.html

purposes, we are interested in **re.split(pattern, string)**, which splits the string on the occurrences of the pattern and **re.findall(pattern, string)**, which returns all the pattern matches within the string.

split()

- reg1 - ' '
- reg2 - `[\s]+`
- reg3 - `\W+`

findall()

- reg4 - `\w+|\S\w*`
- reg5 - `\w+(?:[-']\w+)*|'|[-.(]+|\S\w*`

A simple way to build a tokenizer would be to split into spaces, and we could represent the regular expression simply as ' ', we call this reg1. However, the regular expression does not include tabs or multiple spaces, so we define reg2 with the help of `\s`, we can match any whitespace character, and with + multiple white space characters can be handled. Furthermore, instead of only splitting on white spaces, we could also split on every character that is not a word character, for that, we can use `\W`and call it reg3. Alternatively, we could also use the function **findall** to do the opposite, find every character that is a word character, which is represented as `w`.Notice that if we match words (**findall**), a wider range of cases can be included more easily. Reg4 represents a regular expression that preserves punctuation. For example, in "We'll" we might want to preserve the apostrophe so that "ll" does not lose meaning. Reg5 permits word-internal hyphens and apostrophes. The next list has multiple regular expressions that can be used for a tokenizer, and the table 2.1 specifies the output for each of them.

"We'll be testing a string-searching algorithm, re."				
reg1	reg2	reg3	reg4	reg5
' '	"We'll"	'We'	'We'	"We'll"
"We'll"	'be'	'll'	"'ll"	'be'
'be'	'testing'	'be'	'be'	'testing'
'testing'	'a'	'testing'	'testing'	'a'
'a'	'string-searching'	'a'	'a'	'string-searching'
'string-searching'	'algorithm,'	'string'	'string'	'algorithm'
'algorithm,'	're.'	'searching'	'-searching'	','
' '	—	'algorithm'	'algorithm'	're'
're.'	—	're'	','	'.'
'—'	—	"	're'	—
'—'	—	—	'.'	—

Table 2.1: Different approaches to tokenization.

2.1.2 Stemming and Lemmatization

There are multiple forms of a word. For example, in the phrase "he is writing", the essence that should be captured is the act of writing in the sentence "he is writing", rather than who or when it happened. So it makes sense, sometimes, to represent all the words in the example the same way for feeding the algorithm. Stemming's job is to reduce the inflected word to the word's stem, and this stem does not need to be the morphological root of the word. The word 'writ' does not exist, and this could have an impact on processing. Lemmatization aims at returning to the base form of a word, the word's lemma. So, instead of reducing a word like "were" to "wer", a word that does not exist, the lemmatizer knows that "were" belongs to the lemma "be". Wordnet[3] is a lexical database of English and can be used for the purpose of lemmatization. Several options are made available by nltk, such as one of the first stemmers created and still one of today's most used, the Porter stemmer[4]. Snowball[5], also known as Porter 2, is considered to be an improvement over the first one. Lancaster stemmer[6] takes a more aggressive approach, which can significantly reduce the memory necessary to store the tokens but will turn many of the tokens incomprehensible and reduce the distinction even more. The next table 2.2 aims at observing the differences between the different Text Normalization algorithms.

['writes','writing','wrote','were','am','better','corpora']			
snowball	porter	lancaster	lemmatizer
write	write	writ	writes
write	write	writ	write
wrote	wrote	wrot	write
were	were	wer	be
am	am	am	be
better	better	bet	well
corpora	corpora	corpor	corpus

Table 2.2: Different approaches to text normalization.

2.1.3 Feature Extraction / Text Vectorization

One way we can analyze text is by transforming it into meaningful vectors. There are varied ways this can be achieved; some of the major ones will be described. Bag of Words (BoW) is the simplest one and used to be the chosen one some years ago, but can also be the most expensive one in terms of memory if we are not careful. It represents the occurrence of the words in a given document. Term Frequency–Inverse

[3]https://www.nltk.org/_modules/nltk/stem/wordnet.html
[4]https://www.nltk.org/_modules/nltk/stem/porter.html
[5]https://www.nltk.org/_modules/nltk/stem/snowball.html
[6]https://www.nltk.org/_modules/nltk/stem/lancaster.html

Document Frequency (TF-IDF) is usually a better strategy. Term-Frequency (TF) of a given term(t) in a given document(d) and Inverse Document Frequency (IDF) of a given term(t), multiplied they form TF-IDF. In this way, instead of only counting words, TF-IDF gives insight on how vital the token is. For example BoW will have high values for frequent words and can shadow less frequent words. TF-IDF, on the other hand, will re-weight these words so that less frequent words no longer have a handicap. A significant limitation in both BoW and TF-IDF is that they do not consider the order of the words.

$$\text{tf-idf(t, d)} = \text{tf(t, d)} \times \text{idf(t)} \tag{2.1}$$

There are a few options for the formula of TF and IDF, one that is widely used is to consider Term-Frequency only as of the raw count of the term in the document, in order to value words that appear more times in the same document. For Inverse Document Frequency the following equation, in which n represents the number of documents in the collection and df(t) the document frequency of the term(the number of documents in which the term is present). The job of IDF is to value words that appear in fewer documents.

$$\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1 \tag{2.2}$$

Furthermore, we can apply a normalization step using the Euclidian norm described below to the TF-IDF vectors.

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} \tag{2.3}$$

Word2Vec

Word2Vec[9] learns word associations from a large text corpus and produces a vector for each word in its vocabulary(word embedding). Unlike the previous ones, this model behaves in a way such that the vectors inferred from the model are positioned in the space so that words that share a similar meaning or have a relation between them are close to each other. There are two available methods and they are shown in Figure 2.2.

- On one hand, the method Common Bag Of Words (CBOW) tries to predict a word by its context (surrounding words)
- On the other hand, Skip-Gram uses one word to predict its neighbors.

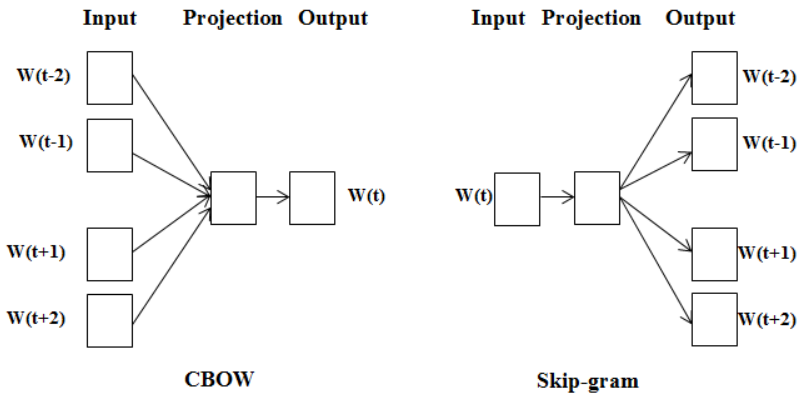

Figure 2.2: Comparison between word2vec models, predicting words with their surrounding words CBOW and predicting surrounding words with a word skip-gram.

Figure 2.3 contains the vectors of countries and their capital cities by a skip-gram model and shows the model's ability to learn relations and organize them. The vectors were projected into a two-dimensional space with the help of a dimensionality reduction technique, Principal Component Analysis (PCA)[10].

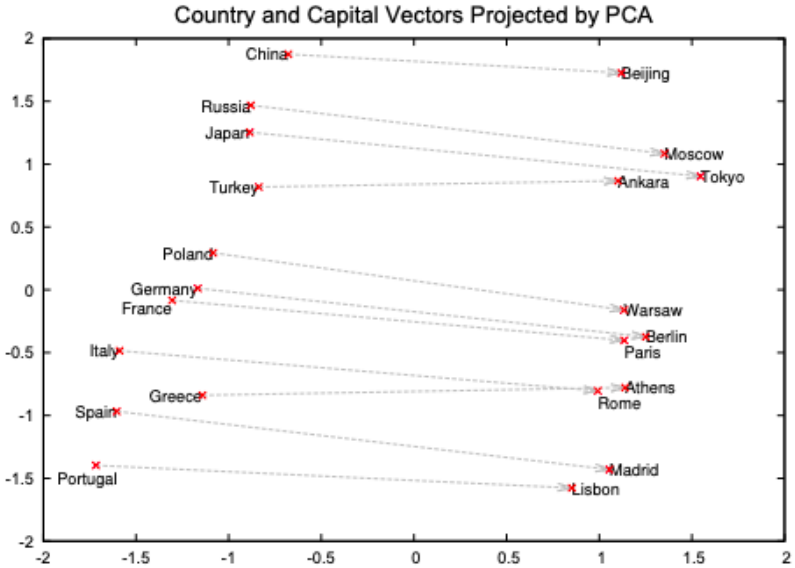

Figure 2.3: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities.[9]

In order to use Word2vec on documents, a strategy has to be defined so that we take into account the different words present in the document. This can be done by summing the vectors, averaging them, or even other weighting strategies where each word contributes differently to the document's final vector, such as TF-IDF.

Doc2Vec

Based on doc2vec[11], Le and Mikolov developed a new way of representing sentences and documents, which is very similar to Word2vec in the strategy used. However, instead of representing word embedding, it can represent whole documents of different sizes with an extra vector, the Paragraph ID, which is document unique. Besides describing the algorithm, the paper also presents sentiment analysis and information retrieval proving to be a better alternative than the BoW and TF-IDF models. Is has the following methods available:

- Distributed Memory (DM) Similarly to the CBOW is used to predict the context of missing words but instead of using just the surrounding words to predict a word has an extra paragraph vector that represents the document.

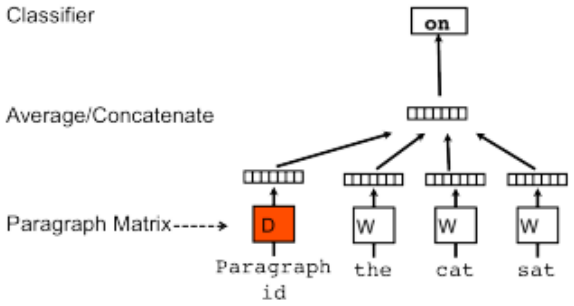

Figure 2.4: DM model.

- Distributed Bag of Words (DBOW) Similar to skip-gram model of Word2Vec, works by predicting words knowing the Paragraph id.

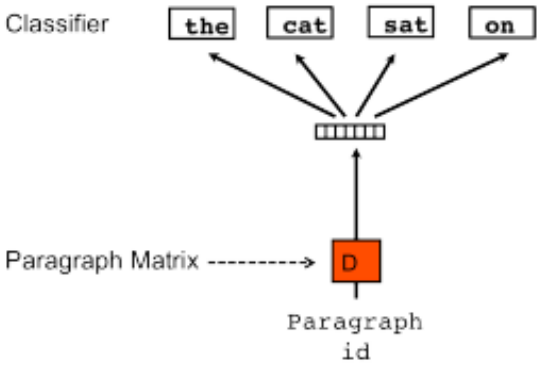

Figure 2.5: DBOW model.

Finally, the output of both models can be concatenated to form a better representation of the paragraph and yield more consistent results overall.

GloVe

After word2vec came out, other methods started being studied that could be an improvement. Pennington, Socher, and Manning argue that a problem with window-based methods, such as word2vec is that they fail at capturing the global context. Global Vectors for Word Representation (GloVe)[12] does not only rely on local context information, but also incorporates global statistics using word-word co-occurrence probabilities that have the potential of encoding some meaning. Take as an example Figure 2.6.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.6: Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus.

The basis of GloVe is that the relationship between two words can be studied by finding the ratio of the co-occurrence probabilities with other words, k. On the one hand, for words k = solid, the probability of ice is higher than steam, and its ratio will be large. On the other hand, for words that are more related to steam, k=gas, the ratio is small. Words that are related to both or neither should be close to one. The objective is to minimize the difference between the dot product and the logarithm of the words probability of co-occurrence. In this way, it associates the ratios with vector differences in the word vector space. In this way, GloVe performs well on the word analogy tasks presented by word2vec. For example king + woman - man = queen.

fastText

Fasttext, is another way of representing word embeddings developed by Facebook's AI Research lab[13]. It is characterized as being fast and efficient. Moreover, it provides a solution to a problem that both word2vec and GloVe fail to address, unknown words. Bojanowski, Grave, Joulin, *et al.* propose a new approach that focuses on the morphology of words. Representing each word as a bag of character n-grams. In which, each character n-gram has a vector associated, and the sum of its n-grams represents a word, so the order does not matter and the vector for unknown words can be calculated as well. Which can then be passed to a skip-gram model in order to predict words.

BERT

Unlike the previous strategies, in which the word representations are context-free, a single vector for each word, Bidirectional Encoder Representations from Transformers

(BERT)[14] is contextual. Which means that the word representation depends on the context it appears. Therefore, it can tackle the polysemy problem, words that are equal but have a different context no longer need to have the same representation. It is bidirectional because both its left and right context is used to predict the masked words. It is not efficient and can take a long time to train, so a pre-trained model is usually imported that can then be fine-tuned for a specific task.

2.2 Machine Learning

One of the most significant areas in computer science today with a prominent future is machine learning. It is a subfield of Artificial Intelligence, which is one of the key emerging technologies, Figure 2.7 shows the results from a survey on the top emerging technologies in customer experience by Gartner in 2019[7]. It is currently being used by big technology companies such as Google, Facebook, and Spotify to provide a more personalized experience to the user through searches, advertisements, songs, videos, and even friendship recommendations. Other areas include image recognition, spam filtering, and traffic predictions. NLP and machine learning are usually tied together. We can use a classifier to classify the vectors calculated in the previous section. Machine learning is usually separated on base of their learning style, either supervised or unsupervised. The difference between them will be explained in conjunction with some algorithms and how they work.

[7]https://www.gartner.com/en/documents/3956088/5-key-emerging-technologies-and-their-impact-on-customer

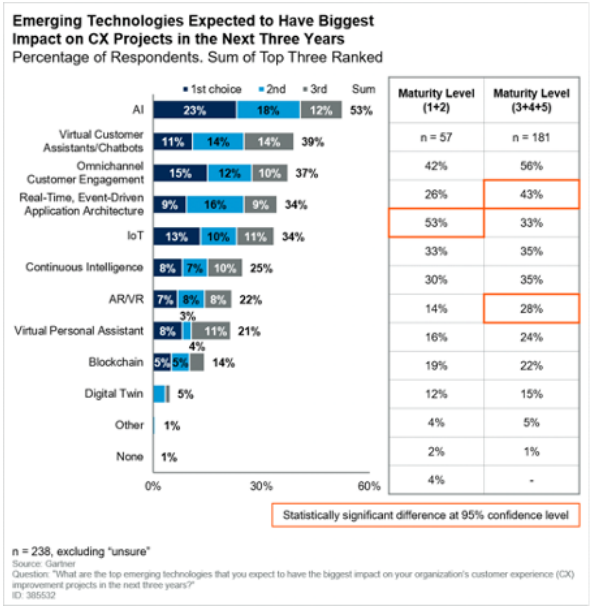

Figure 2.7: Emerging Technologies expected to have the biggest impact on Customer Experience Projects in the next three years according to a Gartner survey made in 2019.

2.2.1 Supervised Learning

Supervised learning refers to the machine learning methods that learn a function that maps the input/features to output/labels using the training set. The function's goal is to predict the correct label for the new input, the testing set. Hold-out is a fast and straightforward technique in which the dataset is split into a training set that is fed to the algorithm for training purposes and a testing set so the model can be evaluated. K-cross-validation is an elegant alternative. With this method, the model will have k training sets to be trained on and k testing sets. The advantage of using k-cross-validation is that a more accurate result of how the model behaves is calculated. Supervised learning can be divided into two categories.

- Classification is the job of taking the input value and assigning it to a class. Binary classification is an example of only two classes.
- Regression is where the output is a continuous number such as a score

Many models have been developed across the years, capable of solving these two problems in machine learning. Artificial Neural Network (ANN) is based on a collection of united nodes, called neurons to resemble a human brain. Boosting algorithms take advantage of multiple weak learners with a weight-related to the algorithm's performance to build a strong one. Examples of such are the famous AdaBoost**Freund1999ASI** proposed by **Freund1999ASI** and the more recent and very fast XGBoost[8]. The Task 1 of eRisk challenge contains labels for classifying the users, which means that it is a supervised

[8]https://github.com/dmlc/xgboost

problem. Therefore, in this section we will be mainly focusing on describing algorithms that are used for supervised learning after presenting a different approach to learning, that does not require any label for the data.

2.2.2 Unsupervised Learning

Unlike supervised learning, there is no labeled data given to the algorithm. The aim here is to find structures or relations between the data. There could be an argument whether to classify Word2vec and Doc2vec as supervised or unsupervised. On the one hand, it can be argued that they do not need any labels to learn from new data. On the other hand, one could argue that it forms its labels using surrounding words, so a more accurate term would be self-supervised learning. A common type of unsupervised learning is clustering, and the aim is to find similar entities and group them. Different ways of grouping algorithms exist; for example, the connectivity-based clustering core idea is that entities are close to related entities and distant from not so related entities.

2.2.3 Models

We will dedicate this subsection to describing the main algorithms used in Machine Learning to solve supervised learning problems.

Linear Regression

Linear regression is the linear approach to estimating real values based on a continuous variable or multiple variables(multiple linear regression). We can simply define it as:

$$y = mx + b \tag{2.4}$$

Where:

- y is the dependent variable
- m is the slope
- x is the independent variable
- b is the intercept

With this, we can represent our data model into a single line and, according to the dependent variables, predict the independent ones. To calculate the slope and intercept that best fit the data, we have to minimize the error between the data points and the predicted value by the linear regression, one way of doing this is with Mean Squared Error. It is one way of calculating the error by taking the sum of the square error difference of every data point divided by the total number of points.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{2.5}$$

Logistic Regression

Logistic regression is used to model the probability of a binary event, each assigned a probability between 0 and 1 with the sum of them being one. Logistic regression, contrary to linear regression is used for classification problems in which the output is discrete and in order to turn the equation of linear regression into a binary problem we can use the sigmoid function, in which z corresponds to the input of the function, for example the equation 2.4.

$$S(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

Support Vector Machines

Support Vector Machine represents data points in space and divides them into separate categories with a hyperplane that should have the largest margin between two classes. Figure 2.8 demonstrates an example of how the model separates the data points. It can happen that the two sets are not linearly separable, and for that case, the inputs can be mapped into a higher dimensional space using the kernel's trick.

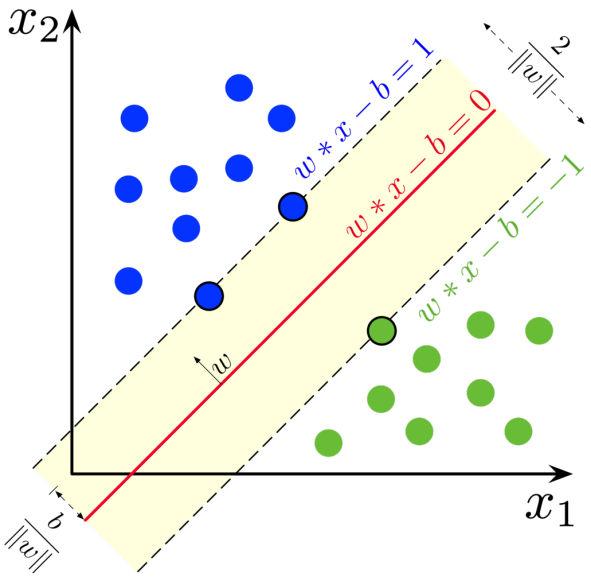

Figure 2.8: Hyperplanes that separate the two classes of a trained SVM model[9].

K-nearest-neighbor

The points are stored and mapped into space, and k closest neighbors will be chosen measured by a distance function, such as the Euclidian distance, that takes the line segment's length between two points or the Manhattan distance, to vote for the class to be predicted. In Figure 2.9, we can see an example, depending on the k chosen, the prediction differs. So it is key to choose the right value to achieve better results. Tuning

would be a way to do it, but a simple solution is to use $k = sqrt(n)$, in which n is the total number of subjects.

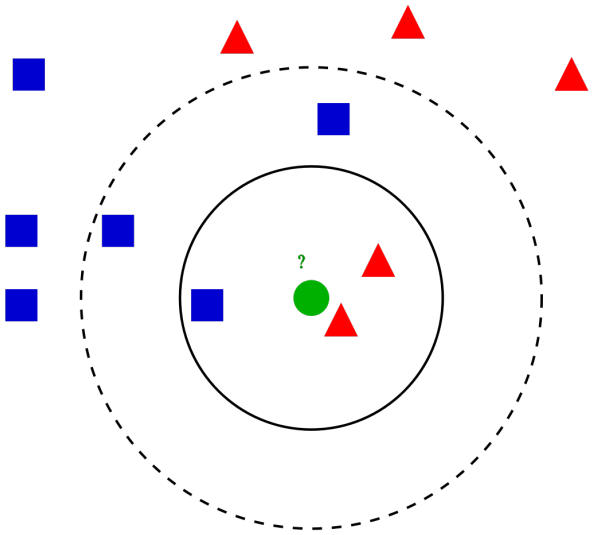

Figure 2.9: Example of k-NN classification. The test sample (green dot) should be classified either as blue squares or as red triangles. If $k = 3$, it is assigned red. However, if $k = 5$, the sample is classified as a blue square[10].

Artificial Neural Network

Artificial neural networks are based on a collection of united nodes, called neurons that resemble the human brain. It's genesis dates back to 1986 when Palm formulated the first artificial neuron[15]. Currently, we define an ANN as a collection of neurons, each connected with a link that has a weight that determines the influence of one node on another. The neurons' output is produced by an activation function that receives the weighted sum of its inputs (weighted by the weight of the connections) plus a bias of one. The neurons are usually organized into layers. One of the first ANN that is still used currently was single-layer and only used for binary classification, it was first published by Rosenblatt, and he called it the Perceptron [16]. Later was found that using two or more layers would be the key to have more significant processing and have the capability to learn non-linear patterns. Deep learning refers to more extensive networks that contain multiple layers between the input and the output layer.

Decision Tree

A decision tree is also a widespread type of model for its intuitive, easiness, and visualization properties. They were used long before machine learning started to be a thing. As the name suggests, they have a tree's shape and has a hierarchy structure

[9]https://en.wikipedia.org/wiki/Support_vector_machine
[10]https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

that represents the importance of the features. The algorithm works by splitting on decisions until the leaves are reached. The output can represent the labels in the case of a classification tree or continuous values in a regression tree case. Random Forest classifier is based on multiple decision trees (bagging), and each tree has a role in voting for a consensus prediction, which will be explained in subsection 2.2.3.

Ensemble Algorithms

Ensemble algorithms take advantage of multiple learning algorithms to achieve better performance than it could have been with any of its constituent algorithms alone. However, it will be necessary more computational power and is advised to use fast algorithms such as decision trees, which Random Forest [17] does. It is a very popular ML algorithm known for its high accuracy and uses bootstrap aggregating, also known as bagging, to combine the predictions from multiple algorithms. Creates many samples of the data and trains a model on each sample, then given a new dataset, we would take the most frequent class or the average if the output is a continuous value. Another technique being used for all sorts of contests is Boosting[18]. The model is incrementally built by adding weak learners that compensate for the misclassified instances. It does this by giving a higher weight to misclassified data and lower weight to correct predictions so that future weak learners can focus on the previous wrong predictions.

2.2.4 Hyperparameter Optimization

Depending on the classifier used, different hyper-parameters are available. One way of performing hyperparameter optimization is with grid search[11]. This method performs an exhaustive search over a specified set of parameters. This strategy to tune the parameters might not be the best in every case. It analyzes the parameters with a brute force technique analyzing every combination of parameters sequentially. Random Search [19] is an alternative to grid-search, and its goal is to be more efficient. Grid-search analyses each combination of hyperparameters available sequentially and therefore explores unpromising regions of the hyperparameter space. Random Search randomly picks a combination to try and, in this way, do a better job at exploring the search space available, reducing the amount of processing necessary to achieve a good result in a sparse hyperspace in a manually defined number of iterations. However, the random search does not consider the scores achieved already to pick a better combination of parameters. Bayesian Optimization [20], on the other hand, in each iteration, picks a sample from the search space based on the optimization of the acquisition function.

[11] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

2.2.5 Evaluation

In the case of binary classification, which is the task we will tackle, several metrics can be defined to perceive how the model behaves on the testing set.

- True Positive (TP) Prediction correctly identifies the label as positive
- True Negative (TN) Prediction correctly identifies the label as negative
- False Positive (FP) Prediction incorrectly identifies the label as positive
- False Negative (FN) Prediction incorrectly identifies the label as negative

- Precision : ratio of correctly predicted positive samples to all predicted positive samples

$$P = \frac{TP}{TP + FP} \tag{2.7}$$

- Recall : ratio of correctly predicted positive samples to all positive samples

$$R = \frac{TP}{TP + FN} \tag{2.8}$$

- F1 : harmonic mean of precision and recall

$$F1 = \frac{2}{P^{-1} + R^{-1}} \tag{2.9}$$

When we are interested in an early classification, such as the case of eRisk challenge, we need to take into account the time frame took to process the decision. ERDE was proposed by Losada and Crestani in [21] and is a metric that aims at evaluating models that take in a series of writings valuing earlier positive predictions and is currently being used as a metric for the eRisk contest task 1.

$$ERDE_o(d, k) = \begin{cases} c_{fp} & \text{if } d = 1 \text{ AND label } = 0 \text{ (FP)} \\ c_{fn} & \text{if } d = 0 \text{ AND label } = 1 \text{ (FN)} \\ lc_o(k) \cdot c_{tp} & \text{if } d = 1 \text{ AND label } = 1 \text{ (TP)} \\ 0 & \text{if } d = 0 \text{ AND label } = 0 \text{ (TN)} \end{cases} \tag{2.10}$$

Setting the parameters depends on the type of application, what can be done in cases where there are a lot of negative cases is setting c_{fn} to 1 and c_{fp} to the proportion of positive cases. lc_o comes to value speed in the prediction and each true positive is assigned a penalty described in the next equation controlled by the parameter o

$$lc_o(k) = 1 - \frac{1}{1 + e^{k-o}} \tag{2.11}$$

ERDE has some drawbacks, such as manually set parameters and not being easily interpretable and a new metric was proposed in «Measuring the Latency of Depression Detection in Social Media» [22] called latency-weighted F1 or F1-latency. They define latency of a system as being the median number of posts that

it observes before predicting a user as depressed. And define a penalty defined by the model's time to predict the user. The penalty is 0 if the prediction is made right after the first post and approaches 1 as the number of posts increase. The variable u, represents the number of rounds needed to make a positive prediction for the subject, sys. And p is set to 0.0119. Because we have to adjust p, so that the penalty is 0.5 for the median of posts, which is 93 for the testing set of eRisk.

$$P_{\text{latency}}\left(u, sys\right) = -1 + \frac{2}{1 + e^{-p \cdot (time(u,sys)-1)}} \tag{2.12}$$

In the plot 2.10 we display the penalty along the Y-axis according to the number of rounds needed to predict a positive value in the X-axis.

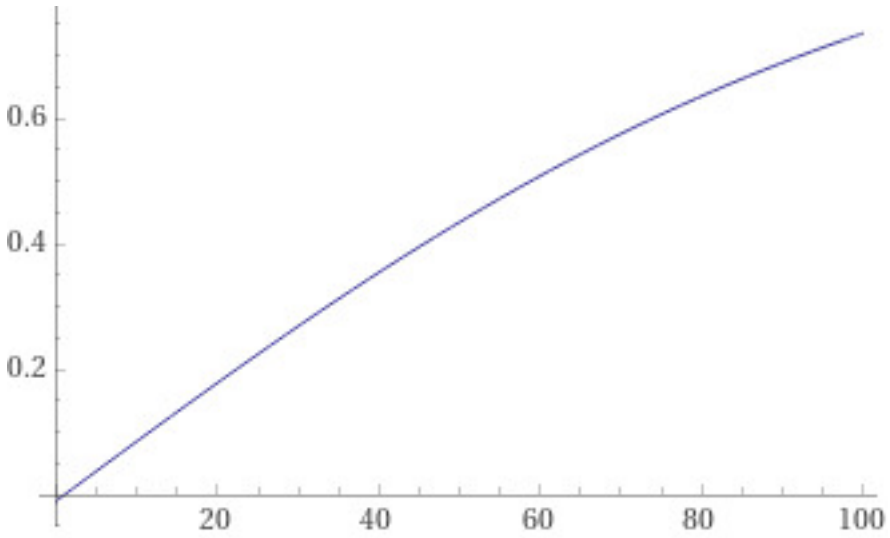

Figure 2.10: Penalty given in an early detection system depending on how many rounds were needed to detect a positive case.

The final metric takes the product of the median of a set of penalties defined before and the model's F1 metric.

$$F_{\text{latency}}\left(U, sys\right) = F_1(U, sys) \cdot \left(1 - \text{median}_{u \in U \wedge ref(u)=+} P_{\text{latency}}\left(u, sys\right)\right) \tag{2.13}$$

2.3 Tool Kits

When dealing with machine learning, some libraries deliver different features. Here are some of the most relevant ones available for Python[12]:

- NLTK[13] is a handy, simple, and good for educational purposes tool kit which has a whole book on the matter[14]. Several pre-processing units are described in the

[12]https://www.python.org/
[13]nltk.org
[14]nltk.org/book

book and aggregating it gives a basis on NLP. It includes multiple tokenizers, namely tweet, which might also be useful for Reddit posts. Also contains options for stemming, including Porter, Snowball, and Lancaster. Lemmatization is also available.

- scikit-learn[15] is a high-level library that contains numerous examples and provides good documentation. It has a wide variety of methods available, ranging from dataset transformations, classifiers, both supervised and unsupervised, which can easily be implemented as well as model selection techniques. It does not offer the possibility of setting the GPU for computation. An exciting feature is Pipeline, which can include several steps, usually a list of transformers and a final estimator.
- TensorFlow[16] is a Low-level and flexible library used to build deep learning models, allows GPU usage by installing the CUDA software[17]. Keras[18] is a library built on top of TensorFlow, which enables a more user-friendly experience.
- gensim[19] is a library that includes implementations of word2vec, fasttext, and doc2vec in python
- NumPy[20] offers multiple mathematical usages for dealing with matrix and vectors.

2.4 Social Monitoring

Social monitoring focuses on analyzing social media to learn about its users and what is happening worldwide. Examples of social monitoring cover a wide range of areas, including political sentiment [23], in which they analyze public opinion, and find a correlation to sentiment measured from Twitter messages. Twitter was also successfully used to predict elections in [24]. Another example is the financial market. In [25], Bollen, Mao, and Zeng study the hypothesis of public mood derived from large-scale Twitter feeds affecting the stock performance of some of the largest companies. If we take a step back and focus on individuals, there have been plenty of studies conducted using Social Media. In [26], Park, Schwartz, Eichstaedt, *et al.* collected data from thousands of Facebook users and built a predictive model for their personality based on their language. Looking closer to the thesis's context and generating health evidence from social data, we find numerous studies on diverse topics. Most disease surveillance systems were developed for influenza before the COVID-19 virus. Back in 2006, a first study introduced the term infodemiology, in which Internet data is used to derive health-related information. In the research, they purchased Google advertisements for

[15] https://scikit-learn.org/stable/
[16] https://www.tensorflow.org/
[17] https://developer.nvidia.com/cuda-zone
[18] https://keras.io/
[19] https://radimrehurek.com/gensim/
[20] https://numpy.org/

specific queries, such as "flu," obtained statistics about the traffic, and demonstrated how these could be useful to predict epidemics. Eventually, new studies explored the potential of utilizing social media and extract behavioral traits, such as emotion and language, to identify disorders in users, such as depression [28]. Twitter [29], Reddit [30] and Facebook [31] were all used to conduct research on the area. Other similar mental illnesses have also been explored during the years. Choudhury et al. studied emotion and behavior changes in new mothers, both in Twitter [32] and Facebook [33]. Similarly, Trifan *et al.* also released research on the topic, intending to identify mothers at risk of postpartum depression [34], which we will further analyze in the next chapter. Depression affects more than 264 million people worldwide and is on the rise[35], it can lead to suicide. [36] addresses this severe topic and shows how social media can influence suicidal behavior and evaluates the risk. Later in 2013, Won, Myung, Song, *et al.* showed that social media might have valuable information for forecasting and preventing suicide. And then, Braithwaite, Giraud-Carrier, West, *et al.* [38] used Twitter to show that machine learning classifiers can identify users at a high risk of suicide. Nowadays, we face the Covid-19 virus, and some research has proved social media to be useful in obtaining insights related to the virus. In [39], Drias and Drias perform sentiment analysis and discover patterns related to the virus.

This year's first task CLEF eRisk[21], "Early Detection of Signs of Self-Harm" was already introduced in 2019. The challenge consists of processing posts sequentially and detecting early traces of self-harm as soon as possible so that the system could be used to social monitor users in a social network, for example. The effort of last year by BioInfo@UAVR combined standard machine learning algorithms with psycholinguistics and behavioral patterns [40]. They used as text vectorization tools BoW and TF-IDF and passed the vectors obtained to a linear Support Vector Machine. Other methods were used, for example, some of the best scores were achieved by Universidad Nacional de San Luis (UNSL)[41]. They opted to use a new model SS3[22] in every task, which can value the relationship between words and categories based on word frequencies. They incrementally build the vector for the subjects using addition, reducing the time to predict the answer as no repeated calculation is required at each iteration.

The same task was proposed for this year and several approaches were submitted. One of the best performing teams, iLab[42], took a modern solution for the problem and trained BERT-based classifiers trained specifically for each task. Another approach was followed by EFE[43], consisting of using Word2Vec representations to build an ensemble of SVM and deep neural networks that take into account both post and user level scores.

[21] https://early.irlab.org/
[22] http://tworld.io/ss3/

2.5 Summary

This chapter intended to give a background on areas useful for extracting, classifying and evaluating health evidence. Starting with pre-processing stages for processing text data, we learn to use regular expressions for turning text into tokens and after apply normalization procedures to produce a less redundant and more uniform vocabulary, stemming and lemmatization. We take a look at how we can represent these tokens or sets of tokens with a set of features, ranging from bags of words based on the importance of the words within the document, to vectors that capture in the space meaningful relationships between them. We focus on learning different supervised learning models, and how we can optimize and use different metrics to evaluate them. Based on the experience and search through out this work, we give an overview on some of the used tool kits in the area, and also explore past works on social monitoring, including approaches took by other teams to solve eRisk tasks.

CHAPTER 3

Methods

In this chapter we try to put in practice what we learned while exploring the previous chapter and pick up several toolkits to help us build a flow capable of processing and analyzing language. Recalling the Figure from previous chapter 2.1, there are usually three stages, Pre-Processing, Feature Extraction and Modeling. But first we will need a data source, so instead of re-inventing the wheel and crawl some websites for content, we will search for challenges that provide us with datasets. The datasets can be in all sort of formats and we have to be prepared to handle them. Python is the language we will use and it has built in functions for reading multiple formats, from text to XML and JSON. After extracting the text from the original source a number of steps is usually applied for Text Normalization. After the text is normalized, we will need some feature extraction type of model to organize and represent the information in the shape of vectors. Now that we have a vector representing each entity and also a label, the modeling phase arrives. In which, a machine learning model is trained to find which features are associated with which label and make predictions based on the patterns it found.

3.1 Challenges

In order to deploy the methods, we will attempt at solving some challenges. The challenges were carefully chosen to go into the encounter of the aim of this thesis. Both challenges consist of analyzing text data from social media and extract information that can lead to the detection. In the first challenge we are interested in detecting postpartum depression, and in the second, the detection of users in risk of self-harm. The first one is an initial project to prepare for the second, and all the analysis was made offline with the intent of solving the problem described in «Social Media Mining for Postpartum Depression Prediction»[34]. The second was an effort to continue the

previous year's participation of the BioInfo@UAVR team in the eRisk challenge. It constitutes online participation in which we had to interact with a server to receive an input and respond with output as detailed in the next section. Our methods used will be described in this chapter.

3.2 Datasets

Reddit[1] was the social network chosen to build the datasets for both challenges. For the first one they searched subreddits containing discussions of a given topic, making it suitable to extract data. Using the Reddit API[2], they scraped subreddits such as *r/postpartumdepression* and ended up with 491 positive posts after cleaning. For the control posts, they used a parenting subreddit *r/beyondthebump* so that the posts' context was similar. Each post is associated with a text file.

For the second challenge, the dataset has the same structure as described by Losada and Crestani[6], consisting of an XML file per subject/user. Each file contains the posts or comments in chronological order. This year's task provides a training dataset, and its characteristics are displayed in Table 3.1.

	Self-harm	Control
#subjects	41	299
#submissions	6 927	163 506
avg #posts/subject	169.0	546.8
avg #words/post	24.8	18.8

Table 3.1: Task1 training dataset.

In order to evaluate the testing set we had to interact with a server in a timely manner. The server sends a post for each user and we have to send back an answer, alert or no alert. For the ranking evaluation, we also had to send a score associated with the user's risk of self-harm.

3.3 Postpartum depression prediction

For this challenge we developed a pipeline[3] capable of turning the text data into vectors and classifying them. For transforming into vectors, we use a TF-IDF implementation of Scikit-Learn, *TFIDFVectorizer*[4]. The final step of the pipeline is a classifier. In this case, SGDClassifier[5] was chosen.

[1] http://reddit.com/
[2] https://www.reddit.com/dev/api/
[3] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

After reading the dataset, we end up with two lists of posts, one corresponds to the training posts, and the second one corresponds to the testing posts, with a split ratio of 80/20, 80 percent of the dataset is used for training stage and 20 percent for the testing stage. And also, two lists containing the golden truth of each post. The next step is fitting the training posts so that the classifier can learn and subsequently predict the unlabeled observations, so that we end up with the predictions from our pipeline.

Scikit-learn offers the possibility of evaluating our predictions using multiple metrics, such as f1, precision, and recall.

When we defined the pipeline we simply used the default parameters for *TFidfVectorizer* and *SGDClassifier*. However, using other parameters can yield better results. To tune the parameters of the pipeline, we use Grid Search, where we can define a set of parameters to be cross-validated and choose a metric to get the set of parameters that behaved best on the cross-validation according to the metric. By looking at the documentation of the steps in the pipeline[6,7], we can see that, for example, TF-IDF has the possibility of defining the n-grams to be analyzed and not only unigrams as set by default. SGD also has multiple definable parameters, such as the learning rate and the loss function.

Now we can see which set of parameters achieved the best score and choose these parameters to predict the testing set. GridSearchCV will automatically use the best-found parameters if the parameter refit is true. Grid-search has a downside; it is not efficient. It searches through every combination of parameters exhaustively. So, for the example above in which we choose four parameters to be tuned and three options for each parameter, the total number of possible combinations is 3*3*3*3 = 81, which means that the full time to find the best set of parameters could take almost 81 times more than using only the default parameters or a combination of our choice. The final scores for our pipeline tuned and not tuned are presented in Table 3.2.

Algorithm	Acc	P	R	F1
Pipeline not tuned	0.741	.658	.762	.740
Pipeline tuned	0.832	.755	.720	.794
Best results in the paper[34]	.93	.90	.89	.90

Table 3.2: Evaluation on the testing set.

Even though the results are still lower than the results in the paper, there is a clear advantage of performing hyper-parameter optimization. We were able to increase the score in most of the metrics for the testing set.

[6]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[7]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

3.4 Early Classification

3.4.1 Word2Vec

Mikolov, Sutskever, Chen, *et al.* demonstrated Word2Vec's[9] perception of words by plotting vectors of countries and their respective capital and observing that the model can distinguish between countries and capitals and learn the relations between them, without any supervised information. Now we attempt at doing a similar experiment on the matter of the thesis. We train a Word2Vec (W2V) model with the full dataset of the eRisk 2020 challenge tokenized and stemmed and extract the vectors for some keywords related to the challenge. Now we want to perceive how the model is behaving, and by doing a PCA[8] on the processed data, we can reduce its dimension to only two and project it into a 2d plot. In Figure 3.1 and 3.2 we show the plot for a CBOW and skip-gram models, respectively.

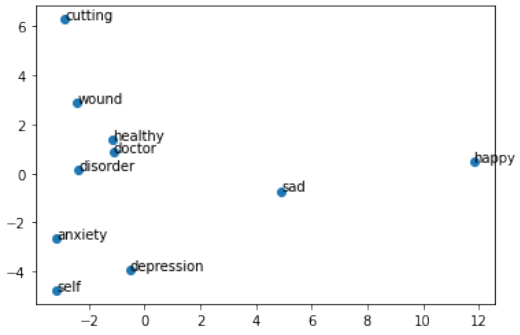

Figure 3.1: Two-dimensional PCA projection of the 100-dimensional CBOW vectors of key words.

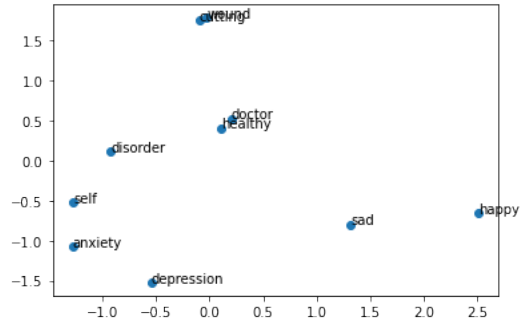

Figure 3.2: Two-dimensional PCA projection of the 100-dimensional Skip-Gram vectors of key words.

Both models seem to organize concepts such that related words are close to each other. Doctor and healthy do not share a similar meaning. However, they are closely coupled together in natural language. Anxiety and depression are close to each other as expected, and interestingly, "self" is also out there, probably due to the tokenizer used separating "self-harm".

3.4.2 Brief Description

Or BioInfo@UAVR approach to the eRisk 2020 problem by Trifan, Salgado, and Oliveira in *BioInfo@UAVR at eRisk 2020: on the use of psycholinguistics features and machine learning for the classification and quantification of mental diseases*[44]. We describe the methods used for Task 1 and Task 2 and present the results obtained. Focusing on the first task, our team BioInfo@UAVR, developed 3 runs to run for the challenge. For all

[8] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

3 runs, we followed a number of common steps in the preprocessing phase. The posts were lowercased and tokenized. Stop words are filtered, based on the stop words list of the Natural Language Toolkit[9]. For the first run we considered BoW and TF-IDF based feature weighting with linear Support Vector Machine with Stochastic Gradient Descent and Passive Aggressive classifiers, the third one is a joint effort in which both the first run and second run were used to make the predictions. Our second run for the challenge was based on a mixture of machine learning and psycholinguistic features. The methodology is composed of five different feature extraction algorithms. The first two algorithms compute features within the data by measuring specific characters' frequencies or words. The first one acts before any processing takes place, and its purpose is to find emojis and punctuation symbols on the given text. The second one receives as argument a list of self-harm related keywords. We extract synonyms and antonyms for every keyword using NLTK's[10] wordnet [45]. Moreover, it also has a collection of sets of words. Some of the sets are absolutist words [46], first-person words, and symptoms. Each set of words is then searched on the data to find its frequency. The third algorithm is a TF-IDF vectorizer, which turns the text into a TF-IDF matrix. The fourth and fifth algorithm use paragraph vectors based on gensim[11] Doc2vec, with two different models, Distributed Memory and Distributed Bag of Words. The output of these five algorithms is concatenated, and the best features are extracted using the transformer SelectKBest[12] from Scikit-learn. These features are then passed to the Adaboost classifier, which led to better results in the validation stage among different classifiers that we trained.

3.4.3 System Architecture

Usually, the strategy to build an algorithm capable of classifying text has three main stages. This concept is generalized in Figure 3.3. The first unit is usually called the pre-processing and is responsible for transforming raw text into a more digestible form to be processed by a machine, the tokens. Now that we have the tokens, we need a strategy to turn the tokens into a vector/features to be interpreted by a machine learning model. This step is called feature extraction, and there are multiple ways this can be achieved. In order to capitalize on various feature extraction techniques, the output of each model is concatenated and fed to a machine learning classifier.

[9] nltk.org
[10] nltk.org
[11] https://radimrehurek.com/gensim/auto_examples/index.html
[12] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

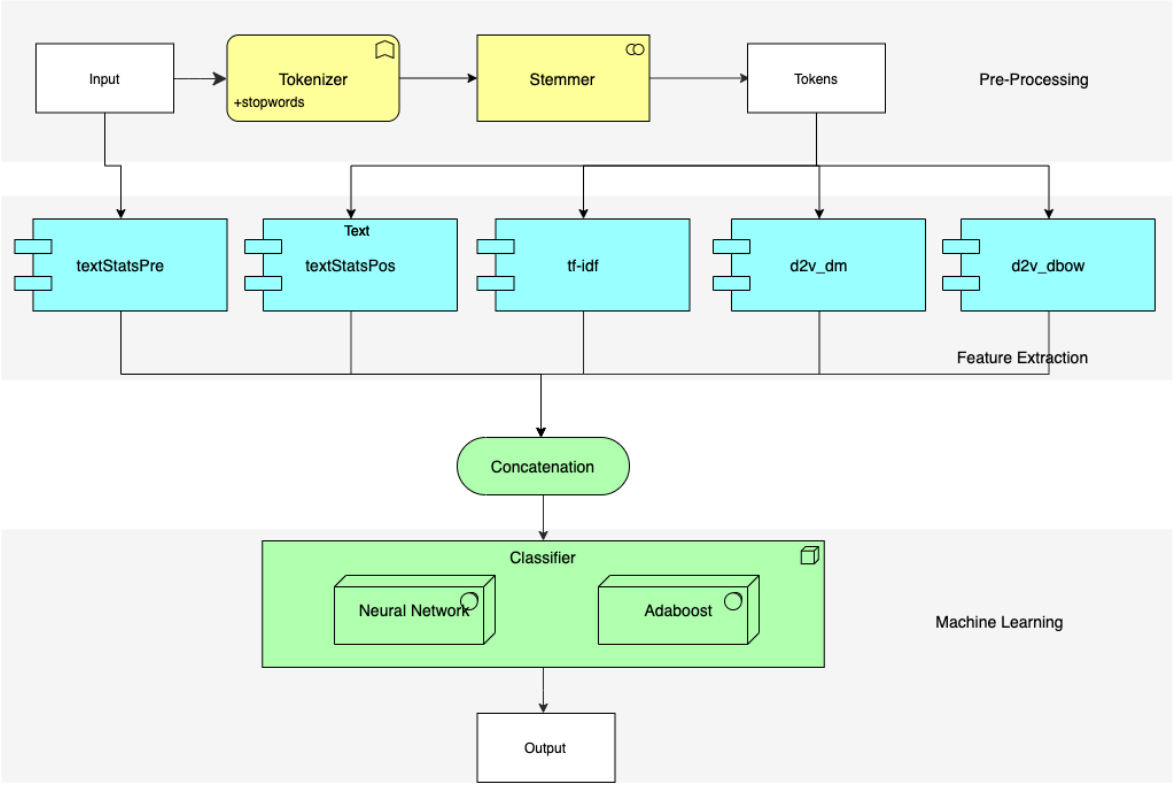

Figure 3.3: Architecture of the system.

3.4.4 Pre-Processing

To have the flexibility of choosing between tokenizers, we built our own class, that has the option for choosing between different tokenization processes. Different options are made available, from *NLTK*[13,14]. Such as a white space tokenizer, which splits simply on spaces, tabs, and newlines. And other more complex ones, with the help of regular expressions, also described in the book. With regular expressions, we specify a set of strings to match and split on them. For example, if we wanted to define the white space tokenizer, which splits on spaces, tabs, and newlines with regular expressions and *split()* we could do it in this way *re.split('[\t\n]+')*. Other regular expressions can be defined that take into account different rules, such as handling hyphenation and punctuation, described in section 2.1.1.

Another essential part of pre-processing that we've discussed earlier is reducing the form of words, and there are also multiple options to choose from here. Such as stemmers, namely Snowball, Porter and Lancaster. And we also set a lemmatizer, that differently from stemmers, aims at returning to the dictionary form of the word, the lemma. We create a class that supports the flexibility of choosing between these various text normalization strategies.

[13]https://www.nltk.org/book/ch03.html
[14]https://www.nltk.org/api/nltk.tokenize.html

3.4.5 Feature Extraction

For the feature extraction, there are already multiple options already discussed in the Background, chapter 2. Besides those algorithms, we also took some inspiration from [47] and built custom components based on a set of psycholinguistic features present in the user writings. These are measured, taking into account the frequency in which they appear in the given text.

Before any pre-processing takes place, we built one model to analyze text, taking into account the different emojis and dividing them into happy emojis, laughing emojis, sad emojis, and surprising emojis, and also the punctuation present in the text, we call this *TextStatsPre*. This class returns a dictionary in which each entry corresponds to the different emojis. To turn it into a vector, we added *DictVectorizer*[15], we also added a scaler to remove the mean, so that the data can behave better when fed to a machine learning algorithm[16]. The flow for this model is displayed in Figure 3.4, text data is turned into an array of features that represent the frequency of certain emojis or punctuation.

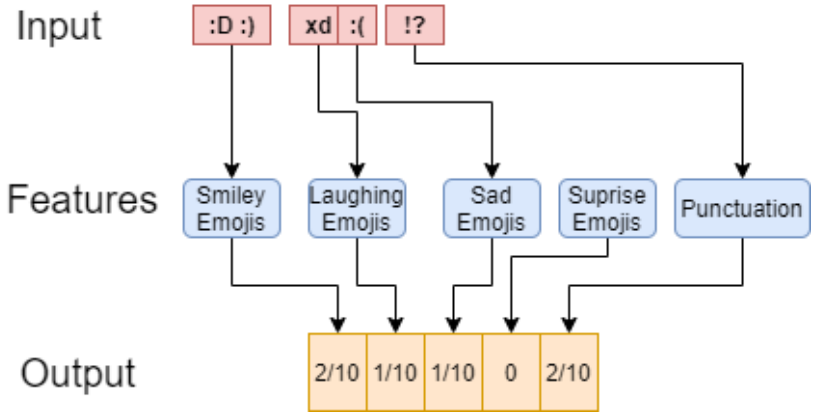

Figure 3.4: Architecture of our first feature extraction model *TextStatsPre*, the features for emojis and punctuation are measured by the frequency of the times they appear in the total length of the post.

Another class was developed to analyze the text after it has been pre-processed and therefore considering only the tokens extracted, called TextStatsPos. It contains some features from [47], such as antidepressants, first-person pronouns, negative words, capitalized words, and absolutist words, displayed in Table 3.3. All the features and their respective sources are listed in Table 3.4.

[15]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html
[16]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Table 3.3: Absolutist words validated by Al-Mosaiwi et al. [46].

absolutely	all	always	complete	completely
constant	constantly	definitely	entire	ever
every	everyone	everything	full	must
never	nothing	total	whole	

The flow for the model *TextStatsPos* is displayed in Figure 3.5. In it we can observe the process that the tokens go through, in order to be turned into an array that represents features.

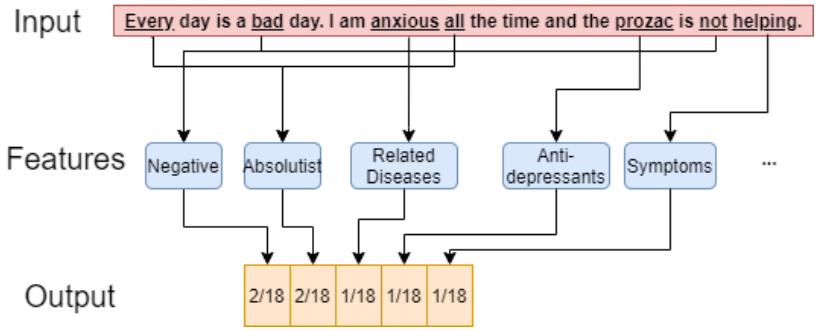

Figure 3.5: Architecture of our first feature extraction model *TextStatsPos*, the features for the sets of words are measured by the frequency of the times they appear in the total number of tokens.

There is also an extra characteristic of this class, which is taking as argument a list of words and searching for its synonyms and antonyms in wordnet[17] and defining them as features as well. These features will be turned into frequencies by dividing their occurrence by the number of tokens. The pipeline for this algorithm contains pre-processing steps, a *DictVectorizer* as TextStatsPos returns a dictionary as well. We also use *VarianceThreshold*[18] to remove features that have no variance.

To feed TextStatsPos with the right words for its features, we used Javascript. By looking at the website's source code, we can build a snippet of code capable of returning the info needed in an array.

[17]https://www.nltk.org/howto/wordnet.html
[18]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

Table 3.4: Linguistic features and source lexica.

Feature	Source
Negative words	`https://www.enchantedlearning.com/wordlist/negativewords.shtml`
Positive words	`https://www.enchantedlearning.com/wordlist/positivewords.shtml`
Symptoms	`https://www.valleybehavioral.com/disorders/self-harm/signs-symptoms-causes/`
Related diseases	`https://www.valleybehavioral.com/disorders/self-harm/signs-symptoms-causes/`
harm lexicon	`https://www.thesaurus.com/browse/harm`
depression lexicon	`https://www.thesaurus.com/browse/depression`
anxiety lexicon	`https://www.thesaurus.com/browse/anxiety`
Antidepressants	`https://www.webmd.com/depression/guide/depression-medications-antidepressants`
Drugs	`https://www.drugabuse.gov/drug-topics/commonly-used-drugs-chartsandothers`

Furthermore, we implement TF-IDF, the previous year strategy of BioInfo@UAVR with the use of *Scikit-learn* implementation *TfidfVectorizer*[19] but using our own defined algorithms for pre-processing instead of the default tokenizer used. The process of using already tokenized text in *TFidfVectorizer* is explained here[20], and consists of using an auxiliary method that simply returns what it receives for the tokenizer and preprocessor options of TfidfVectorizer.

We also implement two versions of doc2vec in order to replicate the Paragraph Vector, DM and DBOW. These will add knowledge to algorithm because the previous feature extraction algorithms did not consider the order of words, but rather only the presence of words. Gensim[48] provides these in python and also facilitates our job by further providing a Scikit-learn interface for Doc2Vec following their conventions, D2VTransformer[21] this means that we can also integrate these in our environment.

Within the dataset, the time of the post is also given. Hence, it can be used to classify the user, considering that positive users tend to write more at late night hours. But what should be the strategy to encode time as a feature? We could take into account only the hours of the post and feed them to the algorithm. However, it wouldn't be a good representation. For example, the discrepancy between writing a post at 1 o'clock and 23 o'clock would be enormous for the algorithm even though they only differ in two hours and should have a similar representation. Ian defines a strategy to

[19]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[20]http://www.davidsbatista.net/blog/2018/02/28/TfidfVectorizer/
[21]https://radimrehurek.com/gensim/sklearn_api/d2vmodel.html

encode cyclical continuous features[22], by using two features deriving a sin transform and a cos transform from the seconds past midnight of the post's timestamp. This turns the features into an interval between -1 and 1. And together they can form a cyclical encoding of time as shown in Figure 3.6.

$$sin\_time = sin(2 * \pi * total\_seconds/seconds\_in\_day)$$
$$cos\_time = cos(2 * \pi * total\_seconds/seconds\_in\_day)$$

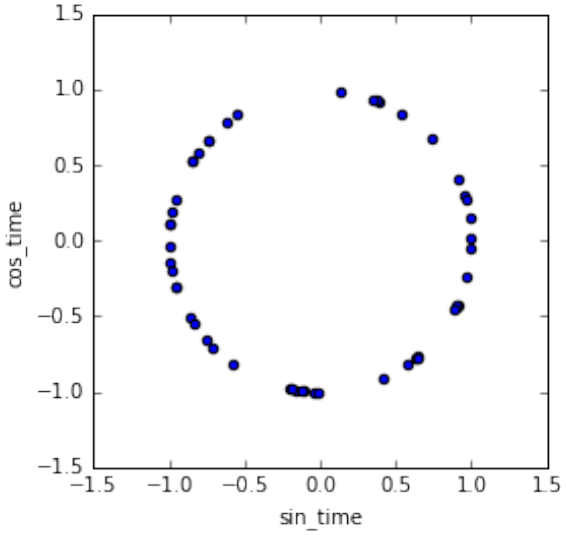

Figure 3.6: Cyclical encoding of time features.

3.4.6 Additional steps

The output from the five algorithms is concatenated, either using NumPy to join the vectors or using Feature Union[23] which applies a list of transformers in parallel to the input data and doing so, we can combine several feature extraction algorithm mechanisms into a single transformer. The concatenation will result in a large number of features. Dimensionality reduction refers to the process of transforming high-dimensional data into low-dimensional data, keeping meaningful properties of the original data. We can apply this to our vast amount of features and use SelectkBest[24], which selects features according to their score on a given function. We opted for $f\_classif$[25]. It computes the ANOVA F-value, which examines if, when we group the numerical feature by the target vector, the means for each group are significantly different for the provided

[22]https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/
[23]https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html
[24]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
[25]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selecti

samples. Measuring in this way the variance of the different features and selecting the ones with higher variance.

3.4.7 Classifiers

Now that we have the vectors, we need to implement a machine learning algorithm that can learn from the inferred vectors and their respective golden truth to predict new vectors without the respective label. Scikit-learn offers the possibility to implement multiple classifiers; Adaboost was the final choice for the online challenge, which seemed to yield the best results; other classifiers tried were Random Forrest and Support Vector Classification (SVC) which is an implementation of SVM.

3.4.8 Validation

As detailed in Subsection 2.2.4, we can perform hyperparameter tuning to achieve better results than we would with the default hyperparameters. To tune them, we will be using Grid Search. With the help of the Parfit library[26], we can parallelize the fit and automatically visualize the results. We define a set of parameters to be tuned for each classifier and pass the training dataset. Then each combination of parameters is cross-validated with $k = 5$, and the scores are plotted over the parameter grid with a color associated that corresponds to the score obtained. In Table 3.5, we show the best score achieved by each classifier with the respective best parameters. This validation section refers to the validation done before running for the challenge. Later in Section 4, we will be doing a more exhaustive tuning that was made after the challenge.

Classifier	Best Parameters	F1 score
Random Forest	max_features: 0.5, min_samples_leaf: 1, n_estimators: 200	0.64
AdaBoost	learning_rate: 0.01, n_estimators: 1000	0.69
Support Vector Classification	C: 100, gamma: 0.001, kernel: linear	0.68

Table 3.5: Best F1 score of different classifiers with a 5-cross-validation strategy on the training set.

Random Forest did a good job in the validation stage and got an F1 score of 0.64. As we can see in Figure 3.7, the middle plot seems to have yielded the best results (dark red) with maximum features of 0.5, which means that each time the algorithm tries to find the best split, it considers half of the total number of features. The best minimum number of leaves is one, which means that the final tree splits if the number of samples in each leaf is not zero. Surprisingly, the best number of trees was 200. We expected the bigger the number, the better the results.

[26] https://github.com/jmcarpenter2/parfit

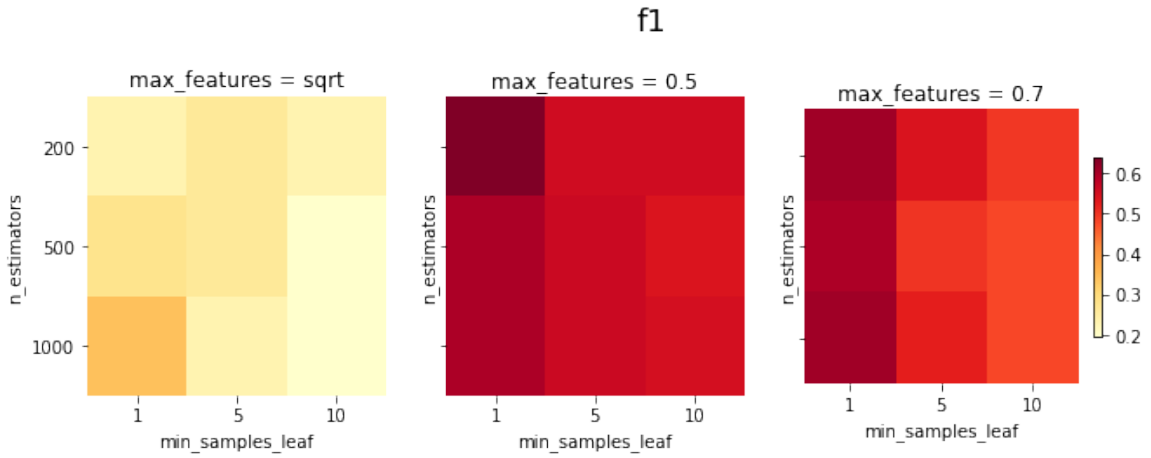

Figure 3.7: Validation for Random Forest using 5-cross-validation.

Adaboost behaved the best in the validation stage, getting the best score with a learning rate of 0.01 and the number of estimators 1000. In this case, a more significant number of estimators seem to do better, as plotted in Figure 3.8. Therefore, this was the configuration chosen to run for the challenge.

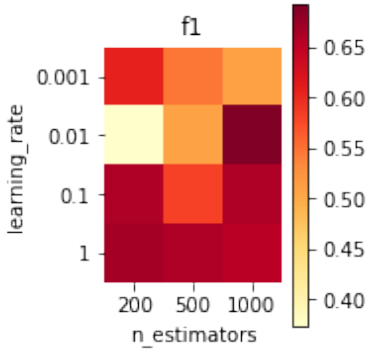

Figure 3.8: Validation for AdaBoost Classifier using 5-cross-validation.

SVC was a close second, the linear kernel did best and was overall more consistent, the best C is 100, meaning we consider a high error for controlling the margin. However, low values of C did ok too. Gamma is not used in a linear kernel, but for the rbf kernel, lower values of gamma seem to be best. The scores are plotted in Figure 3.9.

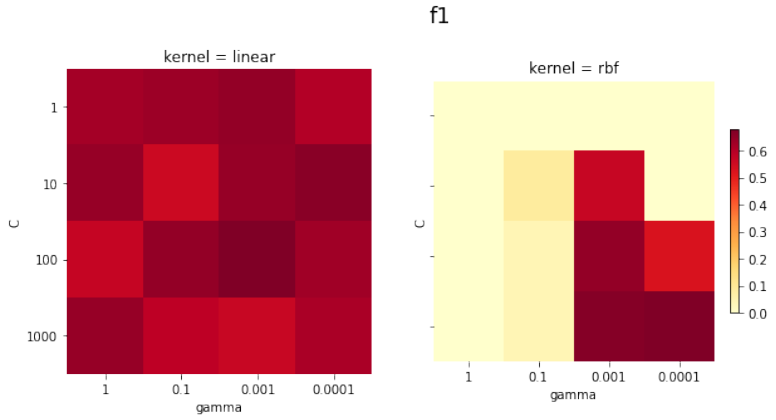

Figure 3.9: Validation for svc using 5-cross-validation.

3.4.9 Testing stage

For the testing stage, we need to interact with a server for a limited period. The server sends us writings, and we have to respond with an answer, 0 or 1, considering that a user is classified as positive can no longer be classified as negative. The following Table 3.6 shows the results of the team BioInfo@UAVR in the first task, "Early Detection of Signs of Self-Harm". Our submission corresponds to run two and achieved the best score in some metrics among our team. And other top results, such as the fourth-best latency-weighted F1 run in fifty-six runs, which is the event's primary metric. The algorithm had some troubles during the live test and had stopped processing posts a quarter way before the end. We will be re-doing the challenge offline with further investigations in the next chapter.

	P	R	F1	ERDE5	ERDE50	latency	speed	latency-weighted F1
Run 1	.609	.375	.464	.260	.178	14	.949	.441
Run 2	.591	**.654**	**.621**	.273	**.120**	**11**	**.961**	**.597**
Run 3	**.629**	.375	.470	**.259**	.177	13	.953	.448

Table 3.6: Evaluation of BioInfo@UAVR's submission in Task 1.

3.5 Summary

This chapter reviewed two NLP challenges. We were successful in constructing algorithms to analyze text and extract information from there. We started to extract linguistic features on text by extracting frequencies of certain words. And then passed to more general methods, TF-IDF and D2V which helped formulating a more knowledgeable prediction. We also tuned the parameters, so that we could achieve the best result possible with the algorithms we have chosen. Multiple plots are displayed that

show the relationship between parameters and score. And we present results for the testing stage, in which we had to interact with a server in a timely manner way, these include standard metrics such as Precision and F1 score, but also metrics that measure early risk detection, ERDE and latency-weighted F1.

CHAPTER 4

Results

This chapter details some offline techniques performed on the eRisk dataset to better analyze our algorithm and build a better version. The model built in the previous chapter performed well on the challenge. There are some time constraints to it, and it can be fully tuned to achieve a better score. The optimization section refers to the experiments done to reduce the processing time and its results. We will also test the different algorithms that compose the main algorithm in Architecture Analysis. Hyper-parameter tuning is done with Grid Search, Random Search and Bayesian Optimization, and we compare them. We will also take a look at the components of some algorithms to analyze feature importance.

4.1 Architecture Analysis

We can take a more in-depth look at the architecture and evaluate each component of the algorithm to understand better how they will behave individually. Both time metrics, measured in seconds, and score metrics are extracted and showed in Table 4.1.

	Time(s)	P	R	F1
Pre-Features	**4**	0.344	0.194	0.22
Pos-Features	403	0.516	0.417	0.442
Tf-idf	174	**0.905**	**0.558**	**0.678**
d2v-dm	123	0.783	0.486	0.581
d2v-dbow	105	0.887	0.461	0.598

Table 4.1: Comparison between the different feature extraction algorithms that compose our final model, classified by AdaBoost. The time refers to the time in seconds took to vectorize the training data in seconds by each model.

Looking at the Table, we see that Pre-Features has a low score and will not help a lot in the final classification. Nevertheless, it is speedy, and we might want to keep it

for the future if we want to maximize the score while keeping a short processing time. Pos-Features achieved an okay score, and is a bit behind state-of-the-art algorithms for feature extraction. It is slow, and we should remove it from our final algorithm or apply dimensionality reduction and diminish the features searched by Pos-Features. TF-IDF did best, achieving very high precision. Furthermore, both doc2vec models did well too. Next, we will look at how we can join these algorithms and take advantage of each of them for our final classifier.

4.1.1 Feature Importance

Some classifiers from scikit-learn have included the method feature importance, it is the case of Random Forest and AdaBoost. It is computed as the total reduction of the criterion brought by a feature, also known as Gini importance, and returns a ndarray, specifying each feature and its importance. The sum of these feature importances is equal to one. We build a pipeline that includes a *TfidfVectorizer*[1] and a *AdaBoostClassifier*[2], and we pass it the full eRisk 2020 dataset. Then we extract the most important features from AdaBoost(ada.feature_importances_) to understand which features are the most relevant ones for classifying the data into positive and negative. We present the top 10 features obtained in Figure 4.1.

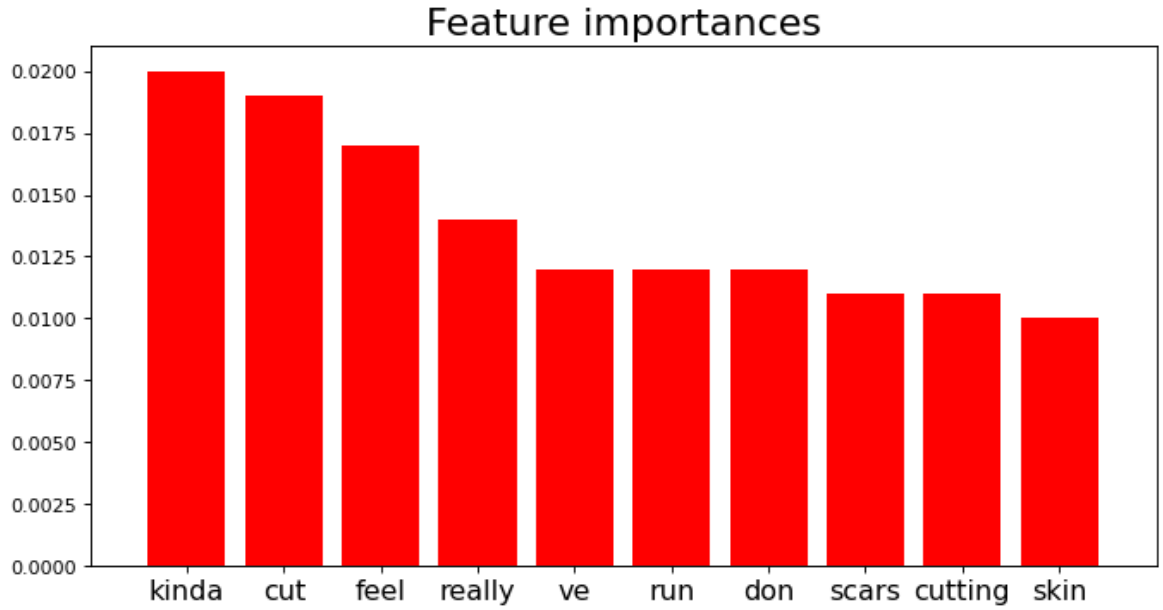

Figure 4.1: Top 10 features from a Feature importance on an AdaBoost classifier that was passed TF-IDF features.

Surprisingly, "kinda" is the top contributing feature. This can be evidence that uncertainty is usually present in self-harm users. Other more obvious words are there,

[1]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

such as "cut","scars" and "skin". As expected, words referring to the individual are present as well, "ve", which before tokenized could be "I've". Other top features include "feel","really","run" and "don".

On the other hand, we can analyze feature importance to perceive better how much each model contributes to the final classification. In this case, we feed the AdaBoost classifier, the output from Pos-Features, TFIDF, and both models of doc2vec. The results are displayed in a bar plot in Figure 4.2

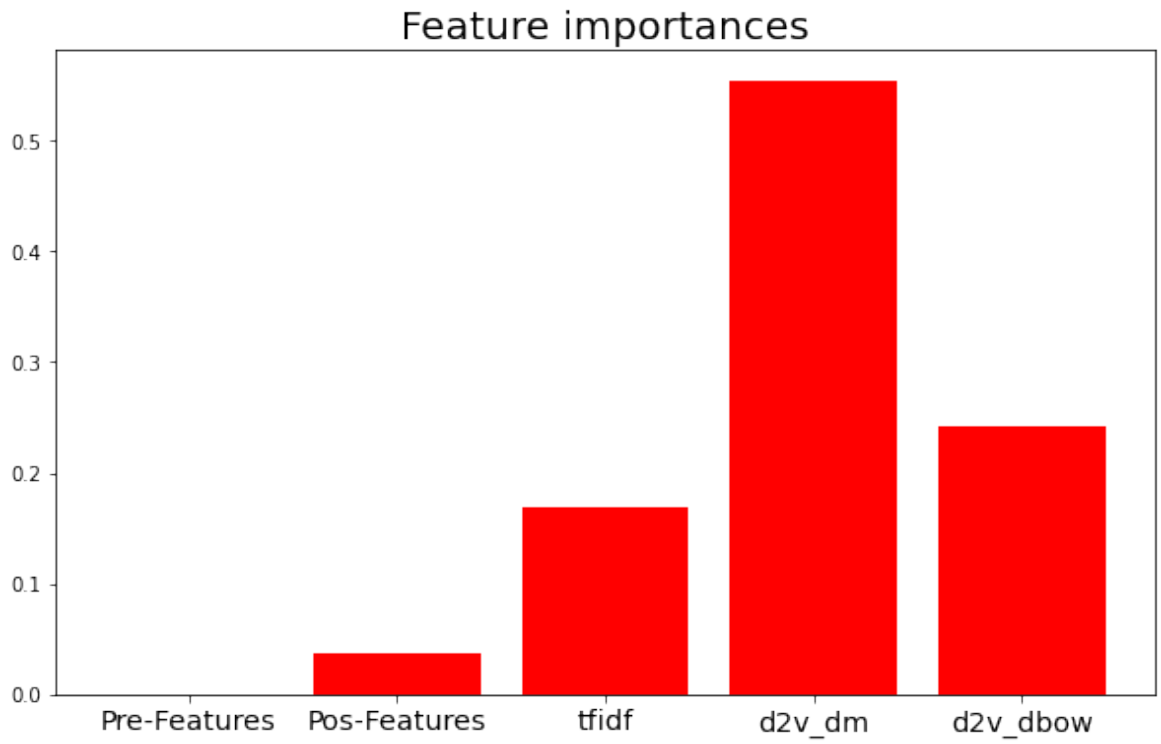

Figure 4.2: Algorithm importance.

The distributed memory model seems to be doing almost half of the work here, while Pos-Features has very little influence on the prediction.

4.2 Optimization

Some steps of the model can be optimized to process data more efficiently, removing unnecessary operations. Our model contains five different algorithms, and the output can be concatenated, making use of the method concatenate from NumPy and is what we call **NPconcat**. If we can set the algorithms to work in parallel, we could reduce the time up to the time of the slowest algorithm. Feature Union[3] offers the possibility of applying a list of transformer objects in parallel to the input data, **FUNP**. In **FUP**

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

besides using feature union, we also set *n_jobs* to one to use all the processors in our computer. Furthermore, we can provide the same normalized tokens to every algorithm and remove pre-features, which has no influence in the prediction as we can see in the Figure of the previous section 4.2, this is called **FUPO**. If we opt to use stop words, time and memory will also reduce a lot, **FUPOSW**. We also added one version that contains Pre-Features, **FUPOSWPre** and one that does not contain Pre-Features neither Pos-Features, **FUPOSW+**.The time took to vectorize the training data is exposed in Table 4.2. The cross validation scores are also exposed to have a term of comparison.

- NPconcat - numpy concatenation
- FUNP - feature union not parallelized
- FUP - feature union parallelized
- FUPO - feature union parallelized and optimized
- FUPOSWPre - feature union parallelized and optimized removing stop words, includes Pre-Features
- FUPOSW - feature union parallelied removing stop words, does not include Pre-Features
- FUPOSW+ - feature union parallelied removing stop words, does not include Pre-Features neither Pos-Features

Model	Time(s)	P	R	F1
NPconcat	920	**0.971**	0.533	0.680
FUNP	912	0.860	0.633	0.720
FUP	543	0.933	0.656	0.768
FUPO	497	0.916	0.606	0.720
FUPOSWPre	377	0.905	0.658	0.761
FUPOSW	329	0.906	**0.681**	**0.773**
FUPOSW+	**263**	0.903	0.653	0.740

Table 4.2: Evaluation of different feature union methodologies, the time is the time took to vectorize the training data in seconds, the cross validation score for precision, recall and F1 is also presented.

In the first three models, nothing is changed in how the algorithm behaves. We only changed the concatenation strategy and processors used. Therefore, the scores are very similar, and the variance might only be due to the algorithms' random factors. In the fourth model, we opt to use the same normalized tokens for every algorithm, and we were able to reduce the time. For the last three models, in which stop words are removed, the score was also good and the best F1 score was even achieved by one of the fastest model, **FUPOSW** which is almost three times faster than the original NumPy

concatenation strategy. It seems that stop words are an excellent feature to have in our final model for time and score purposes. The final pipeline for **FUPOSW** with all the changes is listed in 4.1

Listing 4.1: *Pipeline that receives as input text data, which is tokenized and stemmed. And computes a vector of features obtained with four different feature extraction models, TextStatsPos,Tf-Idf,Doc2Vec-DM,Doc2Vec-DBOW in parallel. The vector is then passed on to the AdaBoost classifier either for training or prediction purposes.*

```
pipeline = Pipeline([
        ('tok', tokenizer(tokenizer_name='pattern1', stopwords=True)),
        ('stemmer', stemmer(stemmer_name='lancaster')),
        ('union', FeatureUnion(
            transformer_list=[
                ('Pos-Features', Pipeline([('posStats', TextStatsPos(keywords)),
                                           ('vectPos', DictVectorizer()),
                                           ('variance', VarianceThreshold()),
                                           ('scaler', StandardScaler(with_mean=False)),
                                           ('select', SelectKBest(f_classif, k=20))])),
                ('tfidf', (
                    TfidfVectorizer(analyzer='word', tokenizer=dummy_fun,
                                    preprocessor=dummy_fun, token_pattern=None,
                                    max_features=300, ngram_range=(1, 2), norm='l1'))),
                ('d2v_dm', D2VTransformer(size=300, min_count=2, dm=1, iter=10,
                alpha=0.025)),
                ('d2v_dbow', D2VTransformer(size=100, min_count=2, dm=0, iter=5,
                alpha=0.01)),
            ], n_jobs=-1
        )),
        ('ada',AdaBoostClassifier(n_estimators=500,n_jobs=-1))
    ])
```

4.3 Hyperparameter tuning

For doing the grid search, it is possible to use feature union as well. Using the best scoring F1 optimization strategy described above, **FUPOSW**, in which we use the same tokenizer and stemmer for the four algorithms, the parameters needed to be searched reduce drastically for the pre-processing unit.

We separate the algorithm into three units for faster tuning. Notice that if we start by tuning the first unit and use standard parameters on the others, then use the already tuned parameters from the first unit, tune the second and leave standard parameters on the third, we will eventually end up tuning the whole algorithm. With this approach, we are reducing the possible combinations. If we assume that each unit has a similar number of combinations of parameters, x, to be tuned, we would have $x + x + x = 3x$ combinations across all algorithms. Diversely, if all the units were tuned together, the number of combinations would equal to $x * x * x = x^3$. This approach, however, has

some drawbacks. It assumes that the units are independent, which is not the case. A pre-processing method such as the tokenizer is going to influence the feature extraction output.

4.3.1 Pre-Processing

For the pre-processing unit, we set up a set of parameters for the tokenizer and the stemmer to be searched. The plot showing both dimensions is represented in 4.3. The strategy to divide the training set was 5-fold. We used the AdaBoost classifier, and the metric used F1.

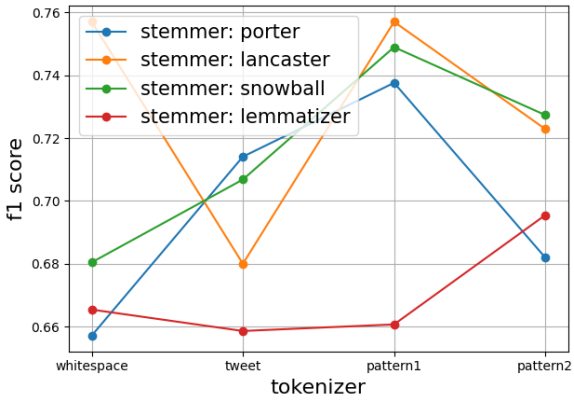

Figure 4.3: Comparison between the different tokenizer and stemmer/lemmatizer algorithms along with F1 scores over training data.

The best tokenizer with more consistent results overall seems to be **pattern1**, defined as `\w+|\S\w*`. It is able to tokenize on white spaces while preserving punctuation marks between words. The best stemmer seems to be Lancaster, which takes the most aggressive approach, reducing the words the most as explored in Section 2.1.2.

4.3.2 Feature Extraction

Let us now analyze each feature extraction algorithm. We can choose if the TF-IDF vectorizer analyzes unigrams uniquely or defines the n-gram to consider and set a maximum for the features to consider and which norm to use. Parameters for the doc2vec model include the learning rate, size of the vectors. Each feature extraction algorithm is tested separately from the others.

As we can see in the plot 4.4, the best score for TF-IDF, displayed in dark red, was obtained by the combination of the following parameters: the maximum number of features is 500, n-gram range is (1, 3). Meaning that having unigrams, bigrams, and trigrams is best. Also, the norm is l1.

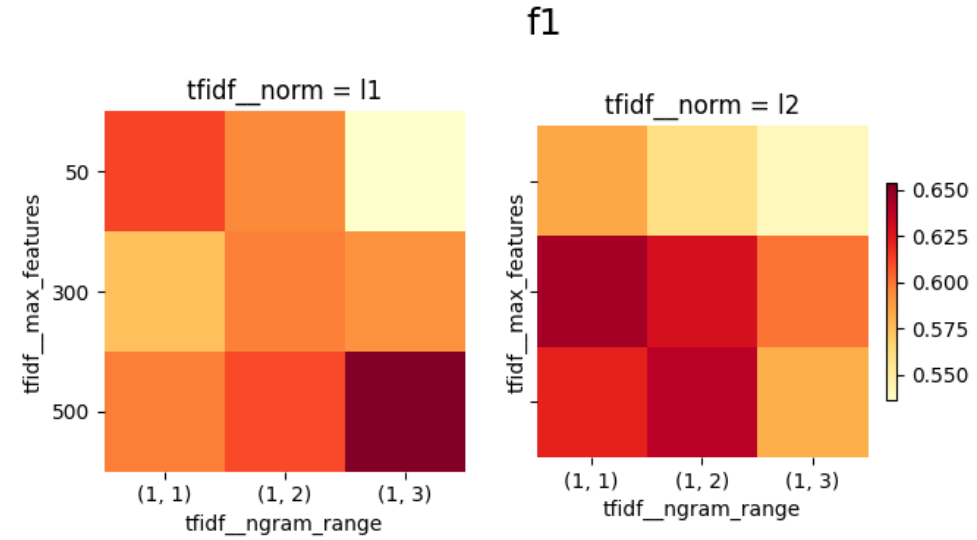

Figure 4.4: Comparison between the different TF-IDF parameters along F1 scores over training data.

The next tuning is going to be on the doc2vec algorithm both on DM and DBOW and its results are presented in the figure 4.5 and 4.6

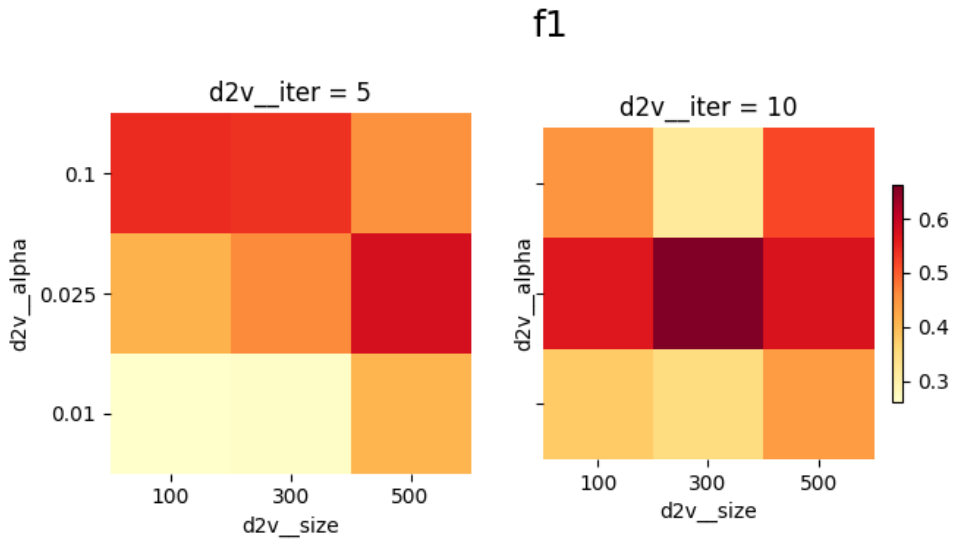

Figure 4.5: Comparison between the different D2V-DM parameters along F1 scores over training data.

The best parameters for the distributed memory version of doc2vec is the number of iterations equal to 10. Alpha equals 0.025 and the vector size 300.

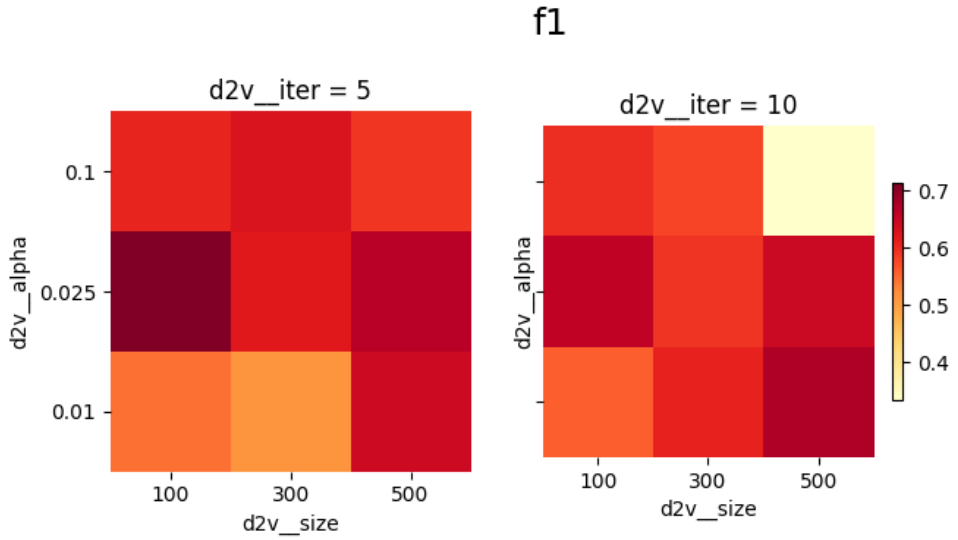

Figure 4.6: Comparison between the different D2V-DBOW parameters along F1 scores over training data.

For the Distributed Bag of Words, the number of iterations is 5, alpha equal to 0.025, and the vector size is smaller for this one, 100.

4.3.3 Classifier

For the classifier tuning, we will be using multiple models to perceive how they behave. A model that has received considerable attention lately in the machine learning world is the Light Gradient Boosting Machine[4] and is efficient, does not use much memory, and is one of the top-performing models used in the Kaggle[5] competitions. Still, to fully take advantage of the model, the right parameters have to be chosen. Some other popular models will be tried, such as Logistic Regression and AdaBoostClassifier.

Firstly we will tune LGBM, defining multiple parameters and their search space. For example, the number of leaves we set the parameter space to be an integer between 3 and 100 with a step of 5. Now we set up three different strategies that differently elect the set of parameters to be evaluated next. Grid search, but instead of using the GridSearchCV from scikit-learn, we define our own that allows us to stop at n iterations. Otherwise, it could take a long time if the parameter space is too large. We also set up RandomizedSearchCV from scikit-learn and BayesianOptimizationCV from scikit-optimize[6], both have the number of iterations adjustable. The results are plotted starting with LightGBM in 4.7, taking into account the F1 score of each tried combination and its iteration. The cross-validation used was 5-fold, and we choose the

[4]https://lightgbm.readthedocs.io/
[5]https://www.kaggle.com/
[6]https://scikit-optimize.github.io/stable/

limit to be 200 iterations. the red dots correspond to the scores obtained by grid-search, the green dots are from randomized search and the blue are from bayesian optimization.

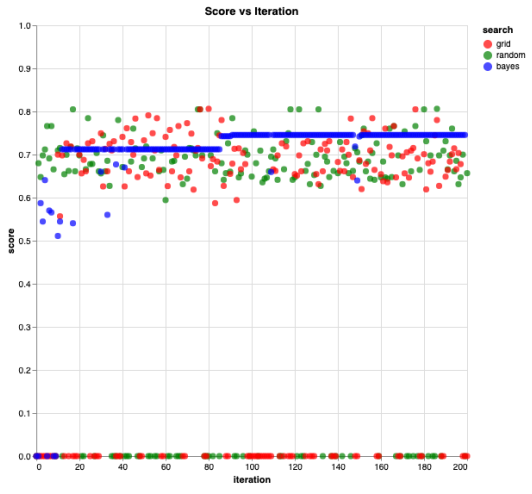

Figure 4.7: LightGBM tuning with 200 iterations

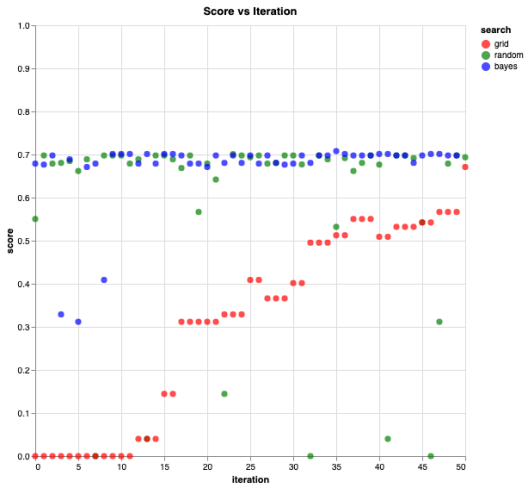

Figure 4.8: LR tuning with 50 iterations.

We can observe in the LightGBM tuning that grid search and random search scored the best results. However, bayes search seems to find acceptable parameters quickly and then stabilizes, seemingly not making use of all the search space available and opting to make small adjustments to the samples, resulting in small variances in the score.

In the Logistic Regression tuning in Figure 4.8, we can see the disadvantage of using grid search. As it runs sequentially, it can try parameters that are not suitable. Therefore it can take a long time to reach a good score. On the other hand, a random search, only with a small number of iterations, is capable of achieving outstanding results. However, it will try out uninteresting samples now and then as it is random. Bayes optimization skips these uninteresting samples, and after learning which parameters are suitable, it does not seem to try out dull samples, being more efficient in that aspect.

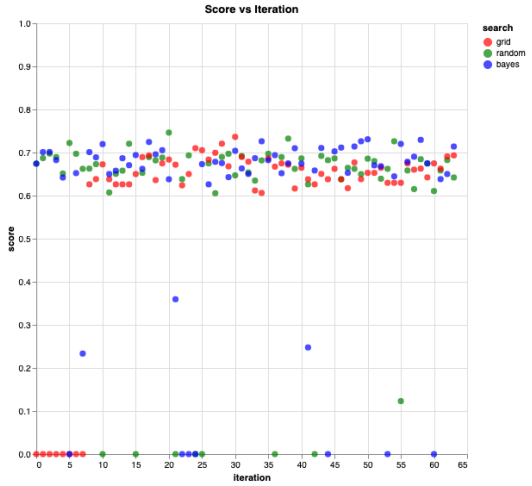

Figure 4.9: DNN tuning with 50 iterations.

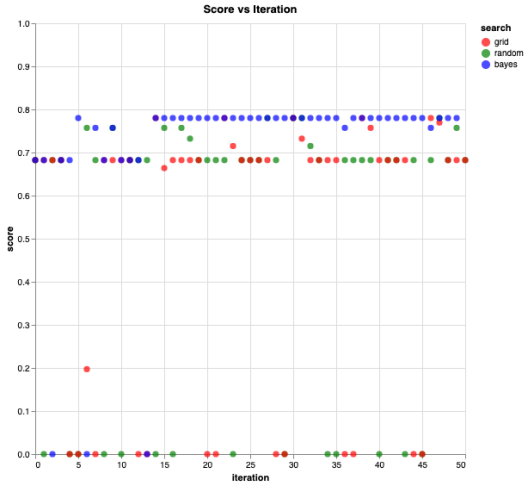

Figure 4.10: SVC tuning with 50 iterations.

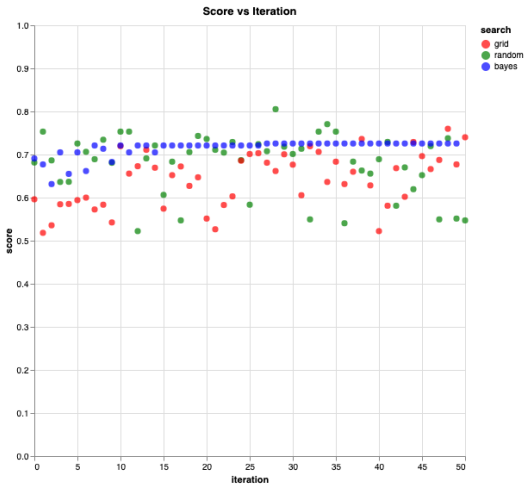

Figure 4.11: Ada tuning with 50 iterations.

We built a Deep Neural Network (DNN) using Keras[7] on top of Tensorflow[8] and set parameters to be tuned, such as the number of neurons in each layer, the activation function, and the number of epochs. The model is sequential, and the input is flattened before entering the input layer. The output layer is of dimension one because the problem in question is a binary one. The random search got the best score. However, it seems that every strategy did well on this classifier.

For the SVC tuning, notice how the scores do not vary that much, this is because some parameters that were tried were actually not used. We set up a linear and a Radial Basis Function (RBF) for the kernel, in order to map the data points into a different space, to be searched and different values for gamma and C, which controls

[7]https://keras.io/
[8]https://www.tensorflow.org/

the error margin for the classifier, to be tuned. However, when the kernel is linear, gamma is not used, and therefore the result will always be the same for the linear kernel with different gamma values. In the case of the nonlinear kernel function, RBF[49], the gamma controls the bias and variance of the models.

AdaBoost, in conjunction with LightGBM, achieved the best cross-validation score, and similarly to LightGBM, random search was the strategy that found the parameters. If the chosen strategy is grid search, then the parameters have to be carefully chosen to reduce the time needed to find a decent result. If we are interested in reducing the time necessary to find a good result, the better choice seems to be bayesian search or random search. In order to find the best score, random search is the recommended choice.

Classifier	Iteration	Search	F1 cross-val-score
LightGBM	46	random	0.805
Logistic Regression	35	bayes	0.708
DNN	20	random	0.746
SVC	14	grid	0.78
AdaBoost	28	random	0.805

Table 4.3: Best F1 score of different classifiers with a 5-cross-validation strategy on the training set.

4.3.4 Parameter Analysis

To better understand the influence of each parameter for the final choice of parameters, we can plot each value tried for each parameter and the score it obtained. The Figure 4.12 shows how each parameter influences the cross-validation score of LGBM. A linear regression is plotted, and the star highlights the best score.

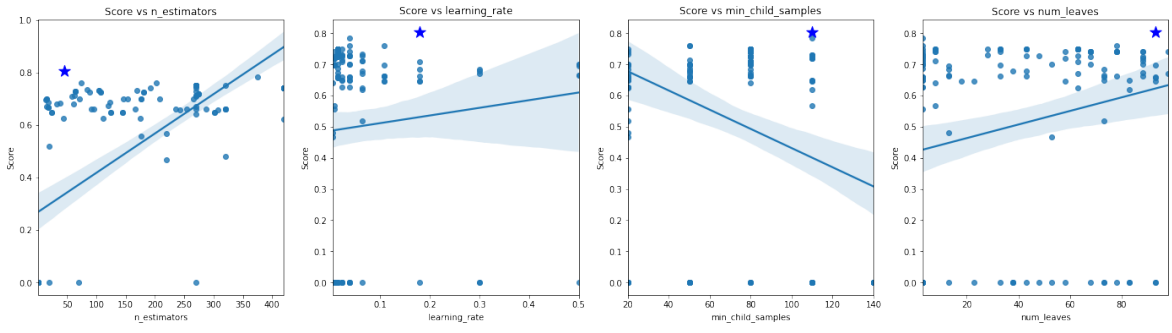

Figure 4.12: Plot parameters of LGBM vs F1 score.

Considering the left plot, we see a linear relationship between the number of estimators used and the score obtained, meaning the more estimators, the higher the score. However, the best number of estimators found was 45. LightGBM has the

possibility of early stopping[9], which means that we did not need to define a specific number of estimators, the model trains until the validation score stops improving for 100 rounds. It is the relation between the parameters that is most important, and we should not rely totally on these individual plots. Therefore, we opt to only show the plots for Light Gradient Boosting Machine. But, we show below the best set of hyper-parameters found for each classifier.

The best parameters found were:

- LightGBM - boosting_type: gbdt, objective: binary, num_leaves: 93, learning_rate: 0.18, min_child_samples: 110, is_unbalance: True, n_estimators:1000
- Logistic Regression - C:10.0, solver: saga
- DNN - optimizer: adam, neurons_l3: 105, neurons_l2: 405, neurons_l1: 205, init: uniform, epochs: 50, batch_size: 35, activation: sigmoid
- SVC - C: 21, gamma: 0.001, kernel: rbf
- AdaBoost - n_estimators: 950, learning_rate: 0.63

4.4 Time-based solutions

Now we try to simulate the eRisk challenge, which is time-based and extract multiple metrics, such as F1 score, F1-latency, and Normalized Discounted Cumulative Gain. For this purpose, we need to define a strategy to handle the processing of posts sequentially. A simple and straightforward strategy would be to process each post corresponding to its iteration. In the first iteration, we would process the first post, and in the second iteration, we would process the second post only. Assuming a time frame of one that corresponds to the average post calculation, the time to process could be described by

$$\sum_{n=1}^{n} 1 = n \tag{4.1}$$

We can also process every post by a given user so that the model can make a more knowledgeable prediction. For example, in the second iteration, we would feed the model the first and second posts. However the time would increase greatly, and can be described as:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{4.2}$$

We can obtain the latency speed up by dividing the latency of each architecture, acknowledging that the execution workload is the same.

$$S_{\text{latency}} = \frac{L_1}{L_2} = \frac{n(n+1)/2}{n} = (n+1)/2 \tag{4.3}$$

[9]https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.early_stopping.html

For a collection of 200 hundred posts per user on average, the result is 100.5. The speed up is 100, meaning the time to process every post could decrease up to 100 times if we choose an incremental strategy. The following strategies are defined:

- Non-Incremental Strategy
  - bruteforce - Computes the vector corresponding to every post(concatenation of posts) read until the current iteration by a given user
- Incremental strategy - computes only the vector for a post at a given iteration
  - individualprob - Takes the probability of the vector
  - avgprob - Averages a list of the probabilities of each vector
  - avgvecs - Takes the average of the vectors computed in each iteration
  - sumvecs - Takes the sum of the vectors computed in each iteration

Now we try the defined strategies in the test set using the AdaBoost classifier, and the F1 latency score is displayed along the number of posts analyzed(iterations) in Figure 4.13.

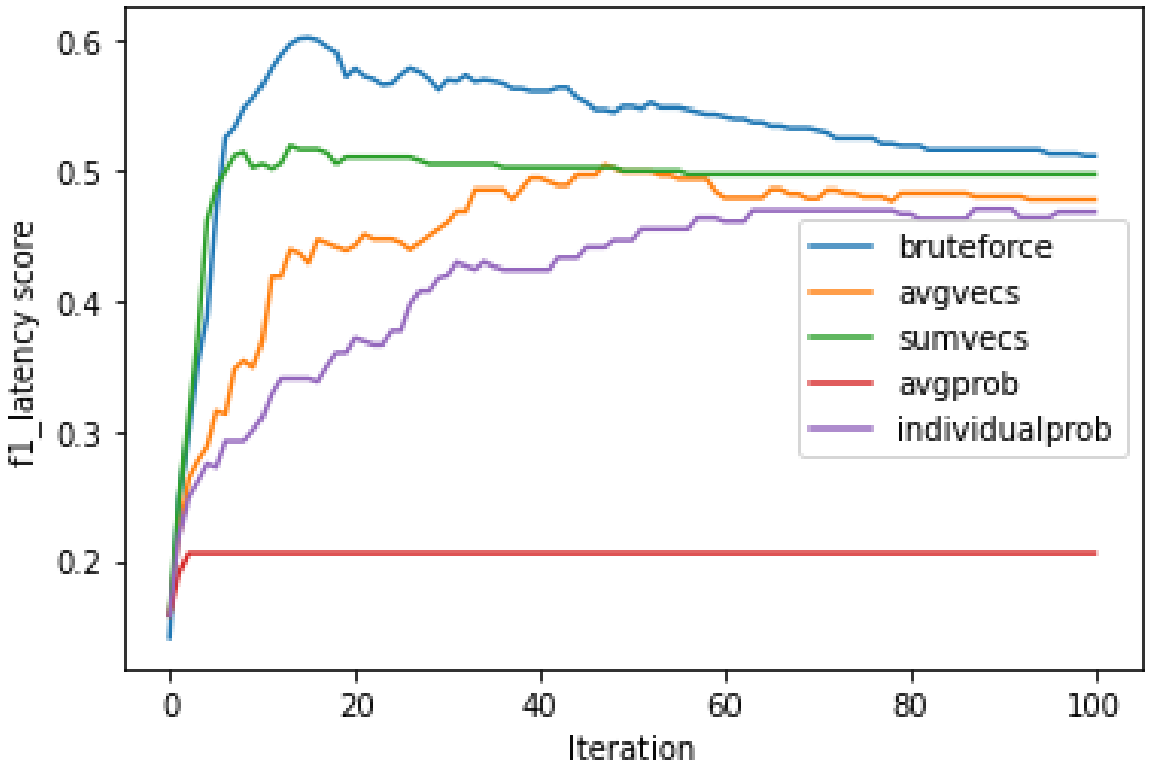

Figure 4.13: Comparison between time-based strategies.

The best results were mostly obtained with **bruteforce** which was also the strategy used for the online challenge. Some of the incremental strategies also did well, **sumvecs** being the best one, even better than averaging the vectors. Notice how the score for

bruteforce quickly goes up almost 0.1 higher then **sumvecs**, but decreases as the iteration passes. While the latter seems to stabilize, which indicates that it might be a better strategy for future work. **IndividualProb** did well, having into account that the prediction is based only in individual writings, without knowledge about other posts, being the fastest out of the five models. **Avgprob**, on the other hand did very poorly, so taking the average of the probabilities is not a good strategy to have.

4.5 Final Test

Now we re-do the challenge using the best parameters found from AdaBoost, LightGBM, and SVC, which did best in the cross-validation score 4.3. Firstly we set each classifier to predict each of the users using all of their posts, on the full dataset. The corresponding scores are exhibited in Table 4.4.

name	P	R	F1
SVC	0.74	0.63	0.68
LGBM	0.71	0.64	0.67
AdaBoost	**0.81**	**0.66**	**0.73**

Table 4.4: Scores on testing set using full dataset.

Now we perform a decision-based approach as described in the eRisk challenge, that consists in processing the posts sequentially and sending an alert when a user is classified as positive. The evaluation should therefore take into account the number of posts needed to make the prediction. Latency-weighted-F1 or F1-latency[22] was designed for this purpose and is an improvement over ERDE[21].

For the early prediction we will use a "bruteforce" strategy, which did best. To fasten up the process, we set up a jump responsible for skipping posts, for example, only analyzing multiples of 10. Which does not matter as we are using a "bruteforce" technique, skipped posts will be evaluated later (processes every post). Notice that if another strategy was used, skipping posts could be problematic. The starting post is set to 10 and 100 on svc100 to boost the F1 measure.

The scores obtained by BioInfo@UAVR are displayed for a better comparison and the run with the best F1 measure, iLab4, and F1-latency, iLab0 were also added to the Tables 4.5 and 4.6. The strategy for the challenge by iLab used BERT-based classifiers, which were then trained specifically for each task. The Table 4.5 shows the decision-based evaluation, in which only the binary decision is taken into consideration.

name	P	R	F1	latency-TP	speed	latency-weighted-F1
SVC100	0.71	0.64	0.68	100	0.47	0.32
LGBM	0.51	**0.76**	0.61	20	0.89	0.54
AdaBoost	0.59	0.68	0.63	20	0.89	0.56
SVC	0.64	0.65	0.64	20	0.89	0.57
BioInfo 0	0.61	0.38	0.46	14	.94	.44
BioInfo 1	0.59	0.65	0.62	11	.96	.60
BioInfo 2	0.63	0.38	0.47	13	.95	.45
iLab 0	0.83	0.58	0.68	**10**	**0.97**	**0.66**
iLab 4	**0.83**	0.69	**0.75**	100	0.63	0.48

Table 4.5: Decision-based evaluation containing scores for precision, recall, F1 and also metrics specific to early classification.

We see that the classifiers did not improve the latency-weighted-F1. But did well and consistent on the metrics. Each classifier being able to reach 0.6 on the F1 metric. LightGBM has a higher recall score, meaning it predicts (as positive) more positive users. SVC does not fail as much in predicting positive users and has a higher precision score.

Another way of evaluating the performance in the challenge is with a ranking-based evaluation. Which takes into consideration a score that represents the user's estimation of risk. And there is a ranking on each writing, capable of handling continuous re-ranking, simulating a real life application. Each ranking can be scored with standard Information Retrieval metrics, such as P@ and NDCG@k, in which the relevance is only binary. The Table 4.6, presents scores for P@10, NDCG@10 and NDCG@100 at 1, 100, 500 and 1000 writings.

	1 Writing			100 Writings			500 Writings			1000 Writings		
name	P@10	NDGC@10	NDGC@100	P@10	NDGC@10	NDGC@100	P@10	NDGC@10	NDGC@100	P@10	NDGC@10	NDGC@100
lgb	0.8	0.67	0.33	0.8	0.69	0.68	0.8	0.68	0.68	**0.8**	0.68	0.68
ada	**0.9**	0.67	0.33	**1**	0.76	0.66	0.9	0.85	0.73	**1**	**0.81**	**0.72**
svc	0.6	0.54	0.31	0.8	0.75	0.65	0.8	0.77	0.67	**0.9**	0.78	0.67
BioInfo 0	0.6	0.62	0.33	0.6	0.62	0.31	0.6	0.62	0.31	-	-	-
BioInfo 1	0.6	0.62	0.33	0	0	0.07	0	0	0.04	-	-	-
BioInfo 2	0.6	0.62	0.33	0.6	0.62	0.31	0.6	0.62	0.31	-	-	-
iLab 0	0.8	**0.88**	**0.63**	**1**	**1**	**0.82**	**1**	**1**	**0.83**	-	-	-
iLab 4	0.8	**0.88**	**0.63**	**1**	**1**	**0.82**	**1**	**1**	0.68	-	-	-

Table 4.6: Ranking-based evaluation containing the scores for Precision at 10, Normalized Discounted Cumulative Gain (NDCG) at 10 and at 100 for different number of writings analyzed.

The results for the ranking-based evaluation were also good, surpassing the previous online results by our team. We also show the evolution of some of the scores for our four algorithms in the next figures. The precision and recall scores are displayed in 4.14,4.15.

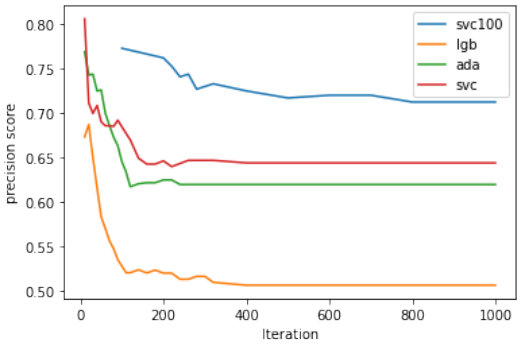

Figure 4.14: Precision score across iteration of posts.

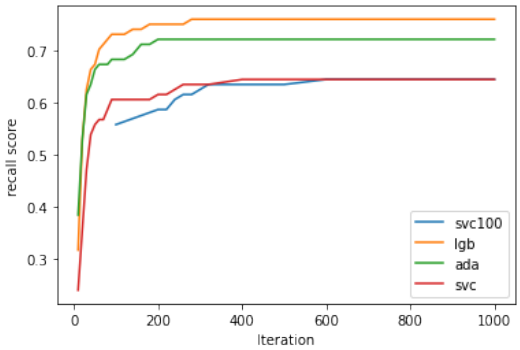

Figure 4.15: Recall score across iteration of posts.

Notice how for SVC100 the precision is higher than SVC, but recall stays the same almost. This is because, SVC100 makes more knowledgeable predictions, starting only to predict at round 100. It will not recall any more true positives, however it will diminish the false positives comparing to SVC. The scores for F1 and F1 latency are displayed in the figures 4.16,4.17

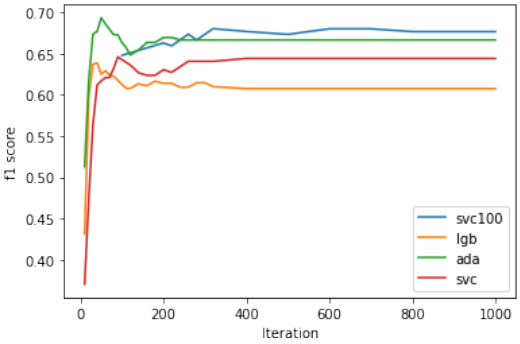

Figure 4.16: F1 score across iteration of posts.

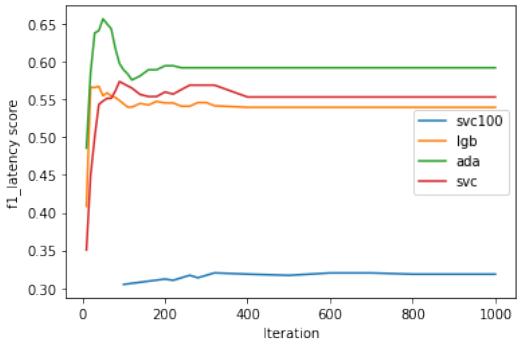

Figure 4.17: F1 latency score across iteration of posts.

We were able to boost the F1 measure for SVC100, but F1 latency score decreased a lot. AdaBoost seems to have performed the best, but is close to svc and lgb and in the long run each of the model is able to capture the relations in the vectors to accurately predict users in risk of self-harm in a time-based manner. The scores for the NDCG@k for k=10 and =100 are displayed in 4.18 and 4.19.

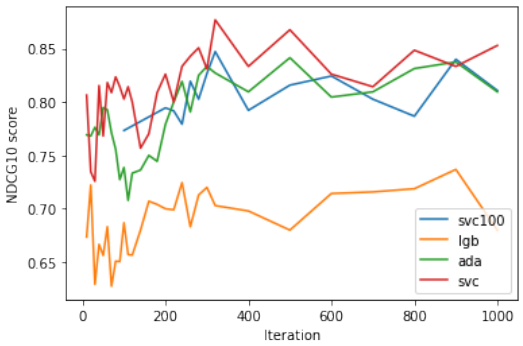

Figure 4.18: Normalized discounted cumulative gain at 10 score across iteration of posts.

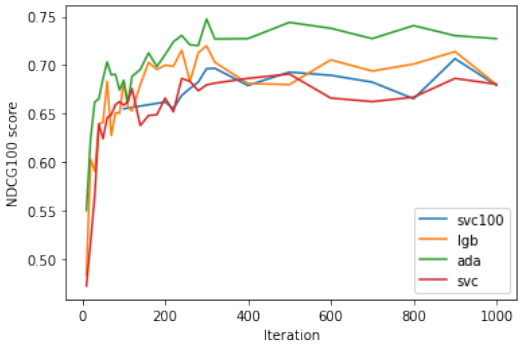

Figure 4.19: Normalized discounted cumulative gain at 100 latency score across iteration of posts.

Depending on the k chosen the score will differ, the higher the k, the less the score is going to be. This metric is useful, for example we can present only the top 10 users that are in risk to an expert if we want higher accuracy but present the top 100 if we want to discover most of the cases.

4.6 Summary

The focus on this chapter was to analyze the algorithm we built and how we could make it better and faster. With the Architecture Analysis we were able to perceive which of our models were performing the best and how much time they took. Then in Feature Importance we learned more about our algorithm and what is the basis for making a prediction. Passing on to Optimization, we were able to keep the score of the original algorithm or even better, and at the same time reduce its latency to one third. On Hyperparameter Tuning we focus on tuning the classifiers as usually is done. But also try to find the best parameters for the other components of our algorithm, pre-processing and feature extraction. Used different tuning strategies other than Grid Search which, as we saw can be limited. We also analyzed our algorithm's behaviour in the conditions of the challenge and how we can improve it.

CHAPTER 5

Conclusion

At the beginning of this work we started by familiarizing ourselves with the tokenization procedures and other text normalization techniques, that are standard tools in almost every NLP project. Then we first approached the problem with a rule-based approach and started finding specific words that were related to the topic and could convey meaning to the classification. Reading the previous experiments in eRisk and some state-of-the-art works on the area, we learn about ways of representing whole documents with vectors that represent the importance of the words in it, BoW,TF-IDF. Word2Vec is able to represent a word with a vector that has meaning and is related in the vector space to other words. Soon, we realized that information about the appearance of certain words is not enough and other methods widely used can handle this already. Doc2Vec scales a level higher and is able to perceive a document's relation to others. Now that we have different ways of representing text, we succeed at taking advantage of multiple models using Feature Union, to build our final representation of the vector . Concluding that a classification that takes into account multiple feature extraction techniques, can be a better alternative then using one alone. Learn different ways of finding the best set of parameters for our models (validation) and how highly important it is to achieve better results. We perceive that with adjustments on the pipeline, such as Stop Word removal and Parallel Computing, there is the possibility of saving time without compromising the score.

We conclude that crawling the social media is a useful method to learn about its users, more concretely understand how they feel and detect mental illnesses with a good accuracy. This information can be an improvement and assistance in the medical system, if we can provide this information to experts. For example like the early prediction challenge, in which we see the evolution of each user in a time-based manner. We released an article[44], describing the approach took to run for the eRisk 2020 challenge,

composed by Trifan, Salgado, and Oliveira.

5.1 Future Work

In the matter of the eRisk challenge, there is the possibility of building an ensemble learning algorithm that takes into account every submitted run for a specific challenge, could this algorithm achieve the best score recorded yet? Imagine a network of medical systems that work together to build weighted predictions of the users.

In order to get the most out of the early classification challenge, a new strategy should be defined that incrementally builds its classification, not having to rely on all the posts at once do the prediction. Summing the vectors of each post should be a strategy to consider, however it does not take into account that the closer the post is to the evaluation, the more important it is. Future work should aim at trying other state of the art models, GloVe measures co-occurrence of words in a global context instead of the local window used by Word2Vec and Doc2Vec. BERT seems to be doing really good and should probably be the focus, it builds a context-dependent, and therefore instance-specific embedding. FastText takes a different approach from every model, stepping in how each word is formed, and captures its meaning by joining the pieces and comparing to other word's pieces.

While working on this thesis, knowledge was obtained, principally around the scikit-learn library. We should explore further other libraries, such as Tensorflow and Pytorch, but keep scikit-learn in our pocket. Future work should also encompass trying new challenges in and out of the scope of eRisk, Kaggle can be a platform to have in mind. Other sources should be explored, other social networks and even platforms that do not have necessarily to be a communication platform, music streaming can also be analyzed.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ISBN: 1558800698.

[2] S. Mansmann, N. U. Rehman, A. Weiler, and M. H. Scholl, «Discovering OLAP dimensions in semi-structured data», *Information Systems*, vol. 44, pp. 120–133, 2014, ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2013.09.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437913001270.

[3] V. Prieto, S. Matos, M. Alvarez, F. Cacheda, and J. Oliveira, «Twitter: A Good Place to Detect Health Conditions», *PloS one*, vol. 9, e86191, Jan. 2014. DOI: 10.1371/journal.pone.0086191.

[4] R. Kraut, M. Patterson, V. Lundmark, S. Kiesler, T. Mukopadhyay, and W. Scherlis, «Internet Paradox: A Social Technology That Reduces Social Involvement and Psychological Well-Being?», *The American psychologist*, vol. 53, pp. 1017–31, Oct. 1998. DOI: 10.1037/0003-066X.53.9.1017.

[5] I. Pantic, A. Damjanovic, J. Todorovic, D. Topalovic, D. Bojovic Jovic, S. Ristic, and S. Pantic, «Association between online social networking and depression in high school students: Behavioral physiology viewpoint», *Psychiatria Danubina*, vol. 24, pp. 90–3, Mar. 2012.

[6] D. E. Losada and F. Crestani, «A test collection for research on depression and language use», in *International Conference of the Cross-Language Evaluation Forum for European Languages*, Springer, 2016, pp. 28–39.

[7] D. Losada, F. Crestani, and J. Parapar, «Overview of eRisk 2019 Early Risk Prediction on the Internet», in. Aug. 2019, pp. 340–357, ISBN: 978-3-030-28576-0. DOI: 10.1007/978-3-030-28577-7_27.

[8] A. M. Turing, «Computing Machinery and Intelligence», *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950, ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. [Online]. Available: https://doi.org/10.1093/mind/LIX.236.433.

[9] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, «Distributed Representations of Words and Phrases and their Compositionality», *Advances in Neural Information Processing Systems*, vol. 26, Oct. 2013.

[10] I. Jolliffe, «Principal Component Analysis», in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_455. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_455.

[11] Q. Le and T. Mikolov, «Distributed representations of sentences and documents», in *International conference on machine learning*, 2014, pp. 1188–1196.

[12] J. Pennington, R. Socher, and C. Manning, «Glove: Global Vectors for Word Representation», vol. 14, Jan. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[13] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, *Enriching Word Vectors with Subword Information*, 2017. arXiv: 1607.04606 [cs.CL].

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[15] G. Palm, «Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity», in *Brain Theory*, G. Palm and A. Aertsen, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 229–230, ISBN: 978-3-642-70911-1.

[16] F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain.», *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[17] L. Breiman, «Random Forests», *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324.

[18] Y. Freund and R. Schapire, «A Short Introduction to Boosting», 1999.

[19] J. Bergstra and Y. Bengio, «Random Search for Hyper-Parameter Optimization», *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html.

[20] J. Snoek, H. Larochelle, and R. P. Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, 2012. arXiv: 1206.2944 [stat.ML].

[21] D. E. Losada and F. Crestani, «A Test Collection for Research on Depression and Language Use», in *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, N. Fuhr, P. Quaresma, T. Gonçalves, B. Larsen, K. Balog, C. Macdonald, L. Cappellato, and N. Ferro, Eds., Cham: Springer International Publishing, 2016, pp. 28–39, ISBN: 978-3-319-44564-9.

[22] F. Sadeque, D. Xu, and S. Bethard, «Measuring the Latency of Depression Detection in Social Media», Feb. 2018, pp. 495–503, ISBN: 978-1-4503-5581-0. DOI: 10.1145/3159652.3159725.

[23] B. O'Connor, R. Balasubramanyan, B. Routledge, and N. Smith, «From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series», vol. 11, Jan. 2010.

[24] J. Digrazia, K. Mckelvey, J. Bollen, and F. Rojas, «More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior», *PloS one*, vol. 8, e79449, Nov. 2013. DOI: 10.1371/journal.pone.0079449.

[25] J. Bollen, H. Mao, and X.-J. Zeng, «Twitter Mood Predicts the Stock Market», *Journal of Computational Science*, vol. 2, Oct. 2010. DOI: 10.1016/j.jocs.2010.12.007.

[26] G. Park, H. Schwartz, J. Eichstaedt, M. Kern, M. Kosinski, D. Stillwell, L. Ungar, and M. Seligman, «Automatic Personality Assessment Through Social Media Language», *Journal of personality and social psychology*, vol. 108, Nov. 2014. DOI: 10.1037/pspp0000020.

[27] G. Eysenbach, «Infodemiology: Tracking Flu-Related Searches on the Web for Syndromic Surveillance», *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, vol. 244, pp. 244–8, Feb. 2006.

[28] M. Gamon, M. Choudhury, S. Counts, and E. Horvitz, «Predicting Depression via Social Media», Jul. 2013.

[29] G. Coppersmith, M. Dredze, and C. Harman, «Quantifying Mental Health Signals in Twitter», Jan. 2014, pp. 51–60. DOI: 10.3115/v1/W14-3207.

[30] M. Choudhury, A. Monroy-Hernández, and G. Mark, «"Narco" Emotions: Affect and Desensitization in Social Media during the Mexican Drug War», *Conference on Human Factors in Computing Systems - Proceedings*, Apr. 2014. DOI: 10.1145/2556288.2557197.

[31] L. Jelenchick, J. Eickhoff, and M. Moreno, «"Facebook Depression?" Social Networking Site Use and Depression in Older Adolescents», *The Journal of adolescent health : official publication of the Society for Adolescent Medicine*, vol. 52, pp. 128–30, Jan. 2013. DOI: 10.1016/j.jadohealth.2012.05.008.

[32] M. Choudhury, S. Counts, and E. Horvitz, «Predicting postpartum changes in emotion and behavior via social media», Apr. 2013, pp. 3267–3276. DOI: 10.1145/2470654.2466447.

[33] M. Choudhury, S. Counts, E. Horvitz, and A. Hoff, «Characterizing and predicting postpartum depression from shared Facebook data», Feb. 2014, pp. 626–638. DOI: 10.1145/2531602.2531675.

[34] A. Trifan, D. Semeraro, J. Drake, R. Bukowski, and J. L. Oliveira, «Social Media Mining for Postpartum Depression Prediction», *Studies in health technology and informatics*, vol. 270, pp. 1391–1392, Jun. 2020, ISSN: 0926-9630. DOI: 10.3233/shti200457. [Online]. Available: https://doi.org/10.3233/SHTI200457.

[35] G. Disease, C. Murray, A. Chattopadhyay, K. L. Chin, S. Lorkowski, and M. Fareed, «Global, regional, and national incidence, prevalence, and years lived with disability for 354 diseases and injuries for 195 countries and territories, 1990–2017: a systematic analysis for the Global Burden of Disease Study 2017», *The Lancet*, vol. 392, pp. 10–16, Nov. 2018.

[36] D. Luxton, J. June, and J. Fairall, «Social Media and Suicide: A Public Health Perspective», *American journal of public health*, vol. 102 Suppl 2, S195–200, Mar. 2012. DOI: 10.2105/AJPH.2011.300608.

[37] H.-H. Won, W. Myung, G.-Y. Song, W.-H. Lee, J.-W. Kim, B. Carroll, and D. K. Kim, «Predicting National Suicide Numbers with Social Media Data», *PloS one*, vol. 8, e61809, Apr. 2013. DOI: 10.1371/journal.pone.0061809.

[38] S. Braithwaite, C. Giraud-Carrier, J. West, M. Barnes, and C. Hanson, «Validating Machine Learning Algorithms for Twitter Data Against Established Measures of Suicidality», *JMIR Mental Health*, vol. 3, e21, May 2016. DOI: 10.2196/mental.4822.

[39] H. Drias and Y. Drias, *Mining Twitter Data on COVID-19 for Sentiment analysis and frequent patterns Discovery*, 2020. DOI: 10.1101/2020.05.08.20090464. [Online]. Available: https://doi.org/10.1101/2020.05.08.20090464.

[40] A. Trifan and J. Oliveira, «BioInfo@UAVR at eRisk 2019: delving into Social Media Texts for the Early Detection of Mental and Food Disorders», in *CLEF*, 2019.

[41] S. G. Burdisso, M. Errecalde, and M. Montes-Y-Gómez, «UNSL at eRisk 2019: a Unified Approach for Anorexia, Self-harm and Depression Detection in Social Media». [Online]. Available: http://tworld.io/ss3.

[42] R. Martínez-Castaño, A. Htait, L. Azzopardi, and Y. Moshfeghi, «Early risk detection of self-harm and depression severity using BERT-based transformers: iLab at CLEF eRisk 2020», English, Early Risk Prediction on the Internet : CLEF workshop, eRisk at CLEF ; Conference date: 22-09-2020 Through 25-09-2020, Sep. 2020. [Online]. Available: https://early.irlab.org/.

[43] H. Bagherzadeh, E. Fazl-Ersi, and A. Vahedian, «Detection of early sign of self-harm on Reddit using multi-level machine», 2020.

[44] A. Trifan, P. Salgado, and J. L. Oliveira, «BioInfo@UAVR at eRisk 2020: on the use of psycholinguistics features and machine learning for the classification and quantification of mental diseases». [Online]. Available: http://ceur-ws.org/Vol-2696/paper_43.pdf.

[45] G. A. Miller, «WordNet: A Lexical Database for English», *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995, ISSN: 0001-0782. DOI: 10.1145/219717.219748. [Online]. Available: https://doi.org/10.1145/219717.219748.

[46] M. Al-Mosaiwi and T. Johnstone, «In an absolute state: Elevated use of absolutist words is a marker specific to anxiety, depression, and suicidal ideation», *Clinical Psychological Science*, p. 2 167 702 617 747 074, 2018.

[47] F. Ramiandrisoa, J. Mothe, F. Benamara, and V. Moriceau, «IRIT at e-Risk 2018», in *9th Conference and Labs of the Evaluation Forum, Living Labs (CLEF 2018)*, Thanks to CEUR editor, a free open-access publication service of Sun SITE Central Europe operated under the umbrella of RWTH Aachen University. This paper appears in vol 2125 Ceur Workshop Proceedings ISSN : 1613-0073 The definitive version is available at http://ceur-ws.org/Vol-2125/paper_102.pdf, Avignon, FR: CEUR-WS : Workshop proceedings, 2018, pp. 1–12. [Online]. Available: `https://oatao.univ-toulouse.fr/22449/`.

[48] R. Řehůřek and P. Sojka, «Software Framework for Topic Modelling with Large Corpora», English, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, `http://is.muni.cz/publication/884893/en`, Valletta, Malta: ELRA, May 2010, pp. 45–50.

[49] A. Bors, «Introduction of the Radial Basis Function (RBF) Networks», in. Feb. 2001, vol. 1, pp. 1–7.