MENTAL HEALTH RISK CLASSIFICATION THROUGH NLP

COMILLAS

UNIVERSIDAD PONTIFICIA

CHARLES GIRARD

INDEX

Abstract

This project aims to create a text processing pipeline in order to detect early risks of depression in Telegram messages from patients. By using labeled text data from the IberLEF data competition, the main goal is to create a supervised classification algorithm that can provide a probability of an individual being at risk of depression. This project will be using text messages sent from 185 spanish-speaking individuals, 100 of which were diagnosed as ‘depressive’ by 10 medical professionals and 85 of which were considered ‘healthy’. Different Natural Language Processing (NLP) techniques along with different Machine learning (ML) classifying models were exhaustively tested in order to find a pipeline that would yield the best accuracy. The best model achieved an accuracy of 88%.

1. Introduction

1.1. Background

The prevalence of mental health issues is on a concerning rise worldwide. Across the globe, there has been a significant increase in the number of individuals grappling with mental health challenges. This growing trend has garnered attention from various organizations, researchers, and governments.

According to a recent report by the World Health Organization, 1 in every 8 people in the world suffer from a mental disorder. Furthermore, the COVID-19 pandemic has greatly increased the global rate of anxiety and depression to more than 26% in just a year. And in 2022, suicide was the fourth leading cause of death among 15-29 year-olds.

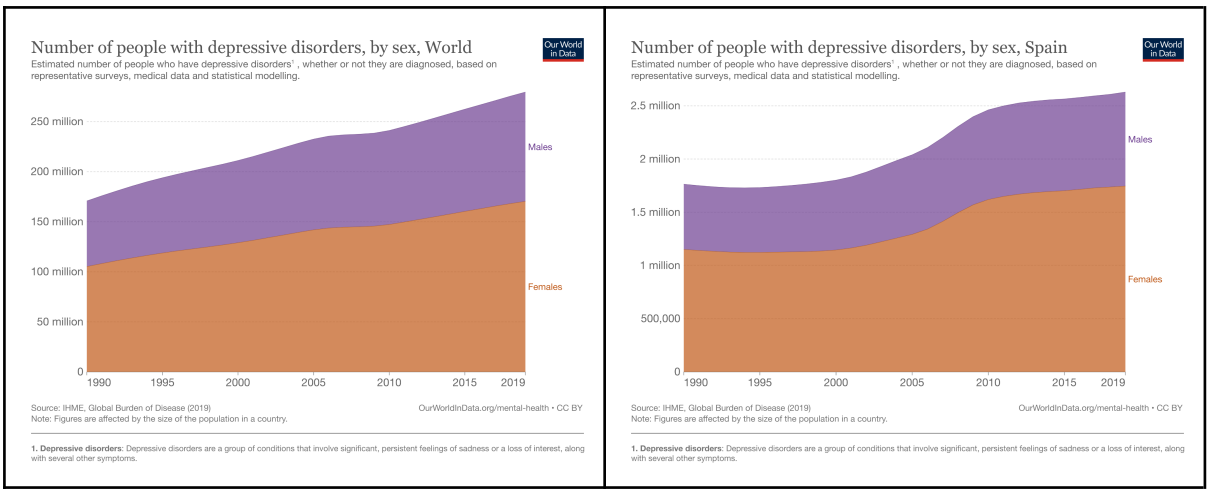


Fig 1. Number of people with depressive disorders in the World and in Spain (1990-2019)

Factors contributing to this surge in mental health issues are diverse and various. The COVID-19 pandemic, as mentioned before, in particular, has played a significant role in increasing mental health problems. The prolonged periods of lockdowns, social isolation, economic uncertainties, and fear of the virus have taken a toll on people's psychological well-being. Studies have shown a marked increase in anxiety, depression, and other mental health disorders as a direct result of the pandemic.

With this information in mind, the early detection of mental health issues such as depression would greatly help to reduce its duration and effect on individuals [1]. It would help the individual's family and friends to plan and to prepare themselves, identify possible treatments and therapies earlier and provide the correct information to both the patient and their surroundings. This information can help the patient find support groups and identify financial supports they might be entitled to.

1.2. Rational

Consequently, there is a growing interest in detecting and identifying mental disorders in social media streams. This answers a demand from society due to the high increase in these problems among the population, in several kinds of mental risks: eating disorders, dysthymia, anxiety, depression, suicidal ideation, and others. Actually, during the last few years, relevant evaluation campaigns like the Cross-Lingual Evaluation Forum (CLEF) have hosted the Early-Risk Identification task (eRisk). However, these campaigns have focused mainly on English, leaving aside other languages, like Spanish. Detecting these risks at an early stage can help greatly reduce the impact of these health conditions by treating them earlier on.

1.3. State of the art

First of all, in order to classify text data, the state of the art method is to use artificial intelligence. More precisely, Natural Language Processing (NLP) techniques will be used as this project deals with unstructured data. Artificial Intelligence (AI) and NLP are also commonly used because they are scalable to large volumes of texts all the while achieving the best performances in overall NLP tasks. In the context of this TFM, the objective performance is precision or accuracy, which is the rate of number of correctly predicted samples relative to the total number of predicted samples.

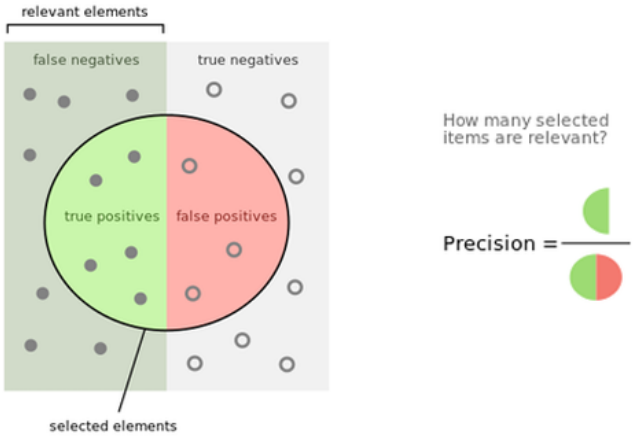


Fig 2. Illustration of precision [2]

While state-of-the-art models in NLP have primarily focused on English, progress in other languages is not far behind. Text classification, the specific task in NLP, is widely studied and numerous freely available models on the internet can classify humor, sarcasm, sentiment, and more. Platforms like huggingface.co have fostered a community of NLP enthusiasts who openly share their work. Over the past two decades, advancements like CNN and RNN models, coupled with accessible hardware like GPUs, have significantly enhanced the training process. The availability of large annotated datasets has also played a vital role, enabling models to learn from diverse examples and generalize effectively to new data. Benchmark datasets, such as the IMDB movie reviews, have further facilitated fair comparisons and evaluations of different text classification models.

In Spanish NLP, one prominent model is the Spanish version of BERT (Bidirectional Encoder Representations from Transformers, discussed more thoroughly later on in this documentation), known as RoBERTa. RoBERTa is a language model pre-trained on a large corpus of Spanish text, which has been fine-tuned for various downstream tasks such as text classification, named entity recognition, sentiment analysis, and more. It has demonstrated strong performance and achieved competitive results across different Spanish NLP applications.

Breakthrough technologies like GPT (Generative Pre-trained Transformer) models have excelled in language generation tasks. Although not specifically evaluated on classification datasets, they have achieved impressive results in tasks like text completion, dialogue generation, and machine translation [3].

Benchmark	GPT-4 Evaluated few-shot	GPT-3.5 Evaluated few-shot	LM SOTA Best external LM evaluated few-shot	SOTA Best external model (includes benchmark-specific training)
MMLU Multiple-choice questions in 57 subjects (professional & academic)	86.4% 5-shot	70.0% 5-shot	70.7% 5-shot U-PaLM	75.2% 5-shot Flan-PaLM
HellaSwag Commonsense reasoning around everyday events	95.3% 10-shot	85.5% 10-shot	84.2% LLAMA (validation set)	85.6% ALUM
AI2 Reasoning Challenge (ARC) Grade-school multiple choice science questions. Challenge-set.	96.3% 25-shot	85.2% 25-shot	84.2% 8-shot PaLM	85.6% ST-MOE
WinoGrande Commonsense reasoning around pronoun resolution	87.5% 5-shot	81.6% 5-shot	84.2% 5-shot PALM	85.6% 5-shot PALM
HumanEval Python coding tasks	67.0% 0-shot	48.1% 0-shot	26.2% 0-shot PaLM	65.8% CodeT + GPT-3.5
DROP (f1 score) Reading comprehension & arithmetic.	80.9 3-shot	64.1 3-shot	70.8 1-shot PaLM	88.4 QDGAT

Fig 3. Benchmark results of different models for various NLP tasks

Regarding the specific task of text classification for detecting depression, some nation-level text classifying tasks have been achieved with ~98% accuracy [4]. Furthermore, in general, text classification has made significant strides, particularly in binary classification tasks. Transformer-based models like BERT, a type of neural network architecture, have achieved remarkable accuracy, surpassing 95% in tasks like IMDB movie reviews classification.

2. Methodology

2.1. Data acquisition

For this TFM, the data will be provided by MentalRiskEs, a novel task on early risk identification of mental disorders through Spanish text messages from Telegram users. In this task, the participants must be able to detect a potential risk as early as possible in a continuous stream of data. MentalRiskEs provided data for eating disorders, depression, and an unknown one which is intended to evaluate the robustness of approaches for new disorders not known a priori. In the context of this TFM, the main focus will be on the depression task.

As mentioned previously, the data comes from a mental risk classification/regression competition organized by IberLEF[5]. It is a shared evaluation campaign of NLP systems in

Iberian languages that has been organized since 2019. It is held as part of the annual conference of the Spanish Society for Natural Language Processing (SEPLN). It aims to promote research in text processing, understanding and generation tasks in at least one of the following Iberian languages: Spanish, Portuguese, Catalan, Basque or Galician.The focus was on one particular mental health issue (depression) that had available data among other labeled possibilities (eating disorder, depression, non-defined disorder). To summarize, the actual data is labeled messages in Telegram groups. Telegram, similar to Whatsapp and other messaging services, allows you to send / receive messages from groups. There exist some spanish-speaking public Telegram groups where the main topic of conversation are mental disorders. This is where the data comes from, that is to say, Telegram users talking about their mental health and how they generally feel. Messages from hundreds of users were extracted and anonymized. Then, 10 medical professionals annotated each individual as being either healthy or disordered. The data contains sensitive information about patients suffering from depression, so any examples shown in this document will be “dummy data”.

For the depression risk classification task, there were 335 patients in total, but due to the nature of the competition, competitors only had access to 185 patients and their labels as the goal of the competition was to use one half for training and evaluation and to predict the labels on the other half.

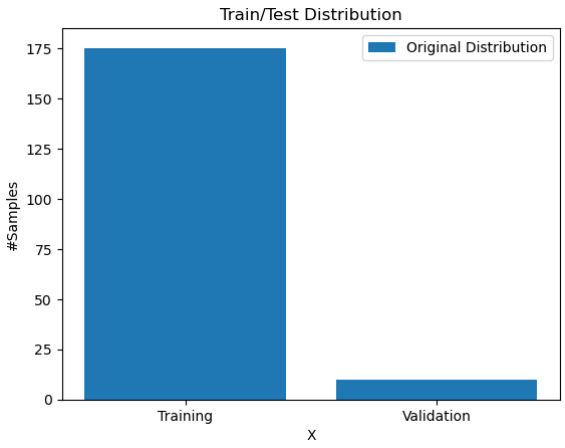
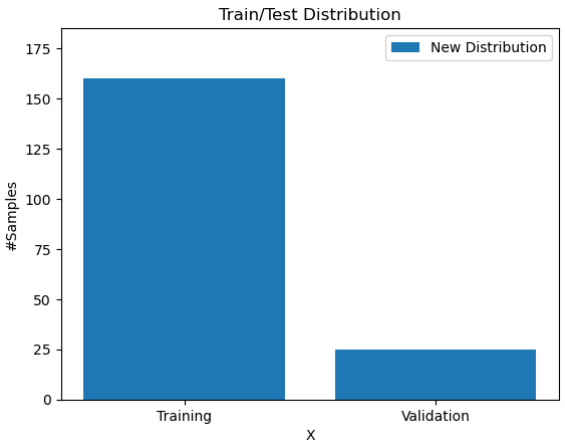
Original Sample distribution	Rebalanced Sample Distribution
	

Fig 4. Re-distribution of the training/testing samples

To summarize, there are messages of 185 patients. Each of these patients is assigned a label “0” or “1” (“1” being classified as suffering from depression, “0” being healthy). For each patient, there are around 40 messages. The aim of the task is to create a process that can successfully predict whether someone is depressed or not based on the messages they have sent.

It is worth commenting on the quantity of data that we have at our disposition. There is a reasonable amount of data for ML classification. We only have 185 “samples”. However, each of these samples contains limited word sequences. This is a limitation for the dataset as standard NLP classification tasks use larger amounts of data compared to this task.

The data was acquired by HTTP request from an organizer’s server using a Python Jupyter Notebook. The rest of the data processing including the development of the classification pipeline will also be done through a Python environment. Python is a programming language of choice for many ML developers for its simplicity/ readability. Furthermore, it has a large ecosystem of libraries and frameworks specifically designed for ML and data science. Popular libraries like NumPy, Pandas, and Matplotlib provide powerful tools for data manipulation, analysis, and visualization. Additionally, frameworks like TensorFlow, PyTorch, and scikit-learn offer extensive functionality for building and deploying ML models. In our case, we will be using these tools all throughout the pipeline. For ML, Python offers comprehensive ML libraries and tools that simplify the development process. TensorFlow, for example, provides GPU acceleration for efficient model training and deployment. Scikit-learn offers a unified interface for various ML algorithms and utilities for data preprocessing, feature selection, and model evaluation.

For computationally simple tasks like data exploration, cleaning and acquisition, the data and the scripts were executed locally on a personal laptop. “Anaconda” was used to manage Python libraries and files. Anaconda is a popular distribution of the Python programming language that provides a package management system, along with a collection of pre-compiled packages and tools, making it easier to set up and manage Python environments. In the context of this TFM, Anaconda allowed for the creation of “virtual environments”. These virtual environments are Python tools that allow for the creation of isolated Python environments that contain specific packages/libraries and their corresponding versions. A virtual environment was created and contained all the necessary Python libraries that were used during this project.

For easy sharing of the current progress of scripts and documentation with the TFM director, as well as for redundancy of the data / scripts, the file sharing platform Google Drive was used. Google Drive is an online cloud file sharing system. Furthermore, when it came time to deploy larger, more powerful models for classification, Google Colaboratory was used. Google Colab is a cloud-based platform provided by Google that allows users to write, run, and share code collaboratively. It is designed primarily for Python programming and data science tasks. It was very useful in the context of this project as it already pre installed useful data science libraries and it allowed for easy installation of new libraries depending on the project’s direction. However, its most useful feature was the free access to computing resources. Code execution in Google Colab happens on virtual machines hosted in the cloud. The resources provided (CPU, GPU, and memory) may have certain limitations and usage restrictions but they are free and are a lot more powerful than what the local computer used during this project was (the computer model was a 2012 Macbook Pro 15”, 8 Gb of RAM, no GPU).

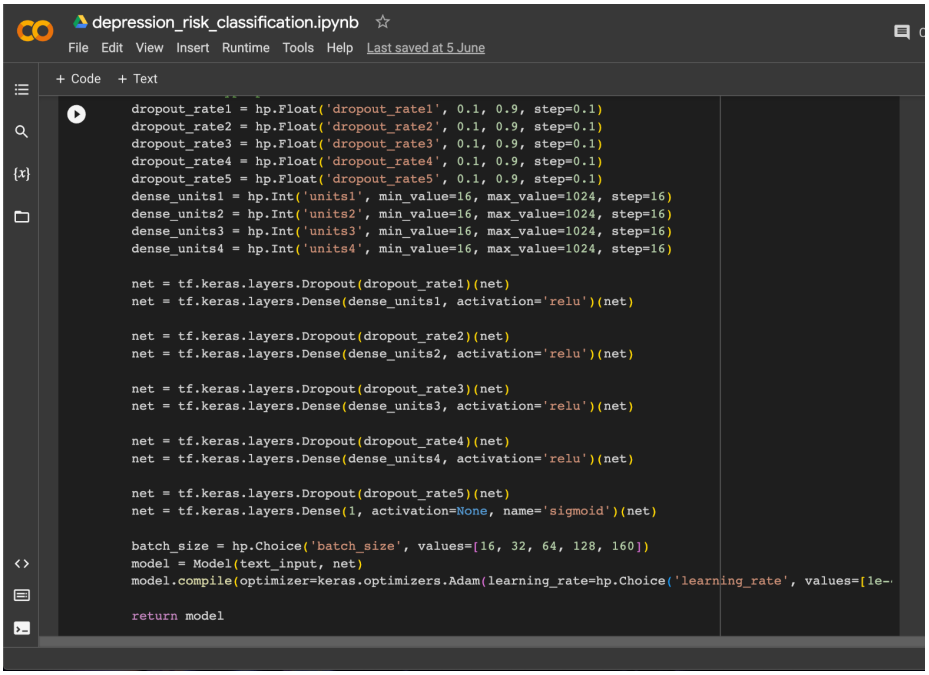


Fig 5. The free Google Collab interface along with the hyperparameter search for a complex BERT model. Collab allows access to a free GPU in order to train ML models

2.1. Exploratory data analysis

To start to get an understanding of the acquired data, one must first start with basic data analytics that will help get insights on how to classify the different patients.

For any NLP task, the first step is to clean (by removing stop words and unwanted tokens) and to stem / lemmatize our data. For this purpose, the Natural Language Toolkit (NLTK) and the Spacy library were used. These are popular Python libraries specifically designed for working with human language data. It provides a wide range of tools and resources for tasks such as text processing, tokenization, stemming, tagging… For lemmatization, stemming and vectorising (all covered in the following paragraphs), the es_core_news_md model was used. This model is a pre-trained model for the Spanish language provided by Spacy. The model has been trained on a large corpus of text and incorporates a range of linguistic features specific to Spanish. The “md” in the model name stands for "medium," indicating that it is a medium-sized model with a balance between accuracy and resource consumption, making it possible to be run locally.

In NLP, excess data that does not add information or insights in the data is often removed as a first step. This is done by filtering out stop words and other artifacts (emojis) for example, among other basic operations. This is tokenization.

Raw message	After basic cleaning
“I LOVE to eat sushi”	“i”, “love”, “eat”, “sushi”

The next step in the text processing pipeline is to stem or to lemmatize the tokens (words).

In NLP, both lemmatization and stemming are techniques used to reduce words to their base or root form. Stemming makes it possible to get the root form of a word by removing the prefix or suffix. (Eg: running -> run). But it does have its limitations, the resulting word may not be a valid word (Eg:relationship -> relatio).

Lemmatization utilizes lexical knowledge resources like dictionaries to perform the word transformation accurately. By considering the context and part of speech, lemmatization can handle irregular words and produce better results. For instance, it can convert "better" to "good" and "running" to "run" based on their usage in the given context.
Lemmatization is more complex but more accurate, so it will be used going forward.

Raw message	After basic cleaning	After lemmatization
“I am eating SUSHI”	“I”, “am”, “eating”, “sushi”	“I”, “be”, “eat”, “sushi”

This step is important to reduce vocabulary size, improve computational efficiency and to help with the performance of ML algorithms.
Now that the data is tokenized and lemmatized, it is possible to start taking a global look at our data.

First of all, looking at the label distribution, there are 100 patients that are suffering from depression and 85 that are not.
Separating them by label and looking at the word distribution gives an insight on how one might classify the patients based on their word usage.

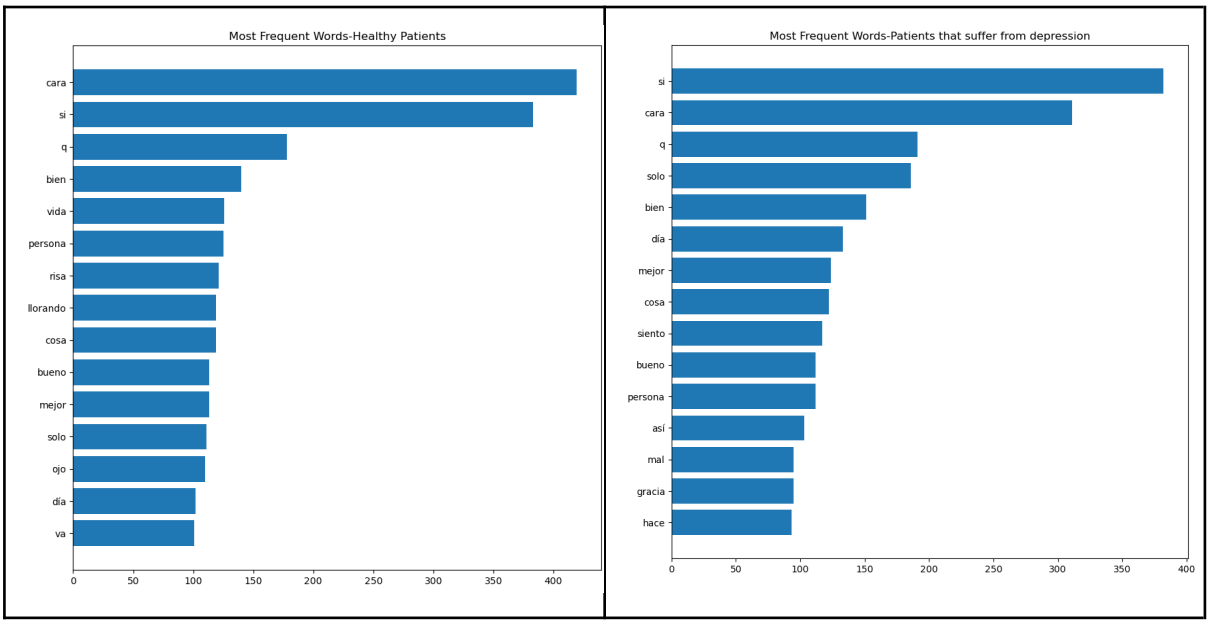


Fig 6. Frequency plot of the 15 most common words per label

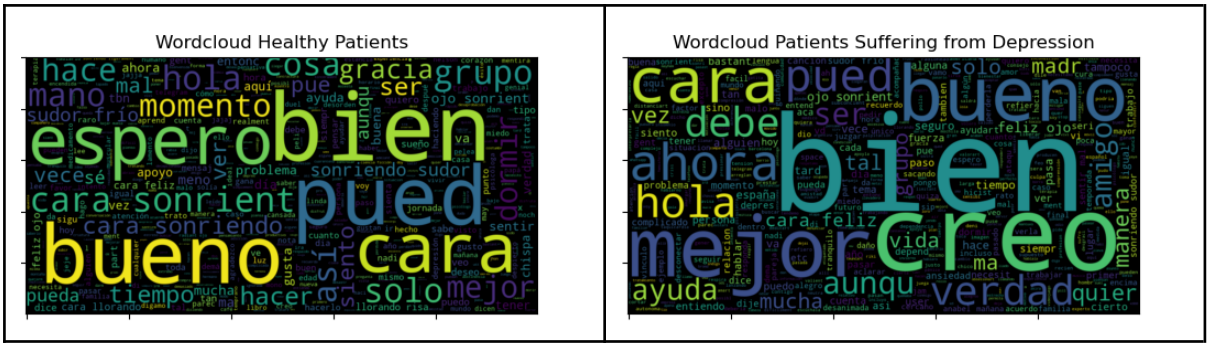


Fig 7. Word Cloud of healthy and unhealthy patients using word count

Seeing the distribution of the most frequent words, we see that healthy patients and patients suffering from depression use similar words (“cara”, “si”, “bien”, “cosa”, “dia”...). This tells us that simply using the words without their context to classify healthy from unhealthy patients will likely be difficult.

However, looking at the frequency of the words is sometimes not enough when there are multiple “documents” (messages). TF-IDF is a widely used technique in NLP that helps to quantify the importance of a term in a document within a larger collection of documents. It aims to highlight terms that are relatively more significant to a particular document compared to the rest of the collection. The TF-IDF score is higher for terms that have high frequency within a document (TF) and are relatively rare across the entire collection of documents (IDF). Therefore, terms that are more unique to a specific document and provide more discriminative power tend to have higher TF-IDF scores.

$$tf\,idf\,(t,\,d,\,D) = tf\,(t,\,d)\;.\;idf\,(t,\,D)$$

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$	$$\mathrm{idf}(t,D) = \log \frac{N}{\lvert\{d \in D : t \in d\}\rvert}$$

Fig 8. The TF IDF formula: “where ft,d is the raw count of a term in a document, i.e., the number of times that term t occurs in document d.” and where N is the total number of documents and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears [6].

Here are the best TF-IDF scores:

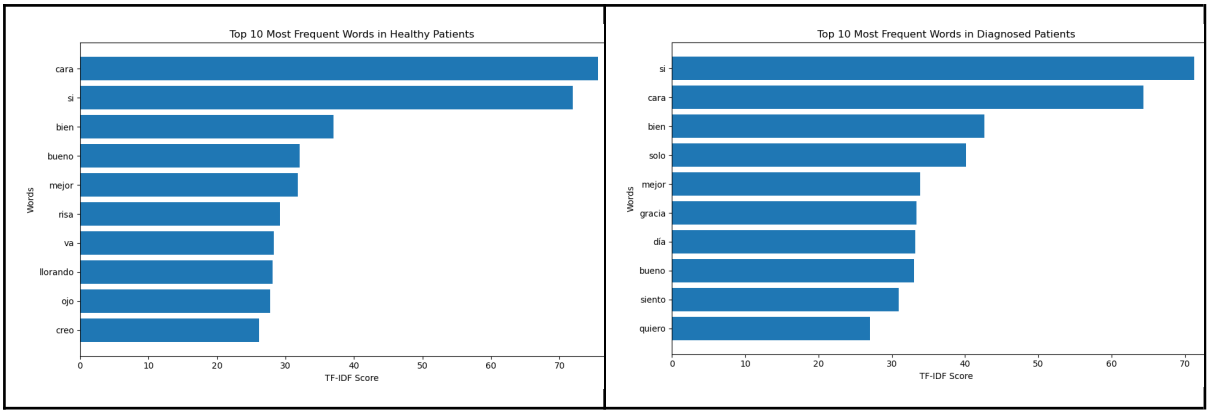


Fig 9. Best TF-IDF scores for healthy and unhealthy patients

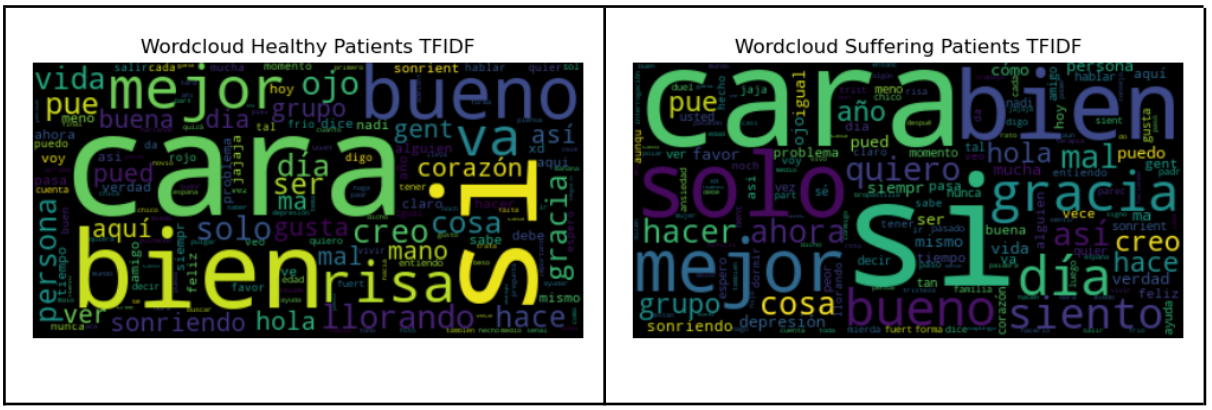


Fig 10. Word Cloud of healthy and unhealthy patient using TF IDF

Once again, even by taking into account the importance of a word relative to its presence in other documents, both the healthy and unhealthy patients share similar words. This once again implies that classifying using only the standalone words will be complicated.
By filtering out the common most frequent words ('bien', 'bueno', 'cara', 'mejor', 'si'), it is possible to start seeing differentiation between the two vocabulary.

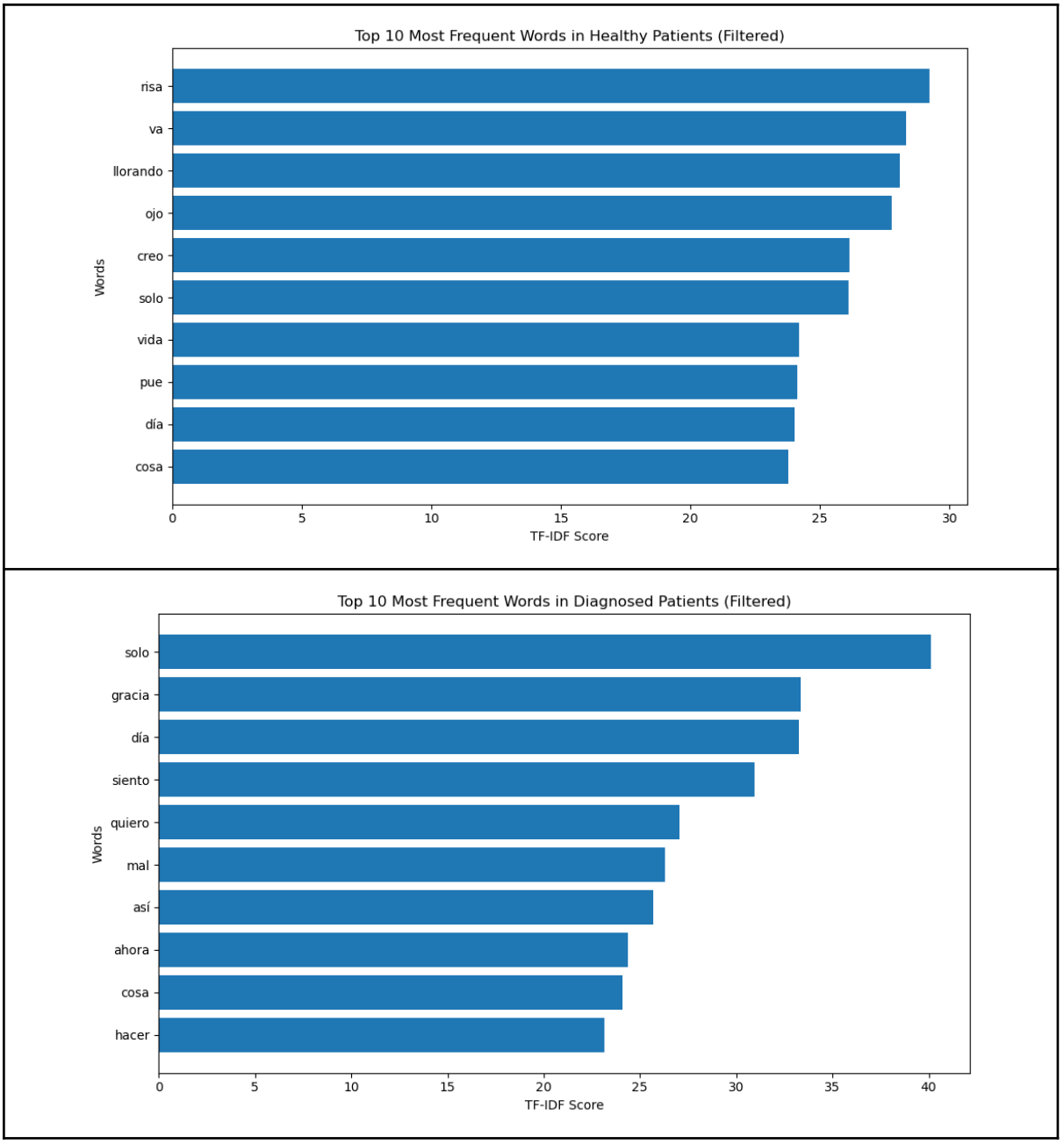


Fig 11. Top TF IDF scores of healthy / unhealthy patient after filtering out the 5 most frequent common words

Some of the most common words after filtering for healthy patients are “risa”, “va”, “llorando”, “ojo” and “creo” while for unhealthy patients, that list is “solo”, “gracia”, “dia”, “siento”, “quiero”. There is a notable difference between the two sets of data. This means that classifying based on the vocabulary used by the different patients could be possible.

2.2 Word embedding and representations

For almost any NLP task, vectorising tokens is a necessity. It consists of translating a word or token into a vector. That resulting vector can then be used in mathematical contexts such

as training data for ML algorithms. By vectorizing the words of healthy and unhealthy patients using the previously mentioned spacy library along with the es_core_news_md model, it might be possible to see distinctions between groups of words used by healthy and unhealthy patients.

First each token is vectorized using the “Spanish Billion Word Corpus” (es_core_news_md) [7], which assigns a numerical vector to each word that represents a semantic meaning. This public model has been trained on a large corpus of Spanish texts in order to be able to give a mathematical representation for each Spanish word. These vectors are all of fixed size (300 in this case but the size is dependent on the embedding model being used). So to visualize them, we must reduce their dimensions down to 2 dimensions. To do this, we will use T-Distributed Stochastic Neighbor Embedding (TSNE) clustering. TSNE clustering is used to represent complex, high-dimensional data in a lower-dimensional space while preserving the local structure and relationships between data points.It basically finds a mapping from the original high-dimensional space to a lower-dimensional space (often 2D or 3D) such that similar data points are modeled as close neighbors in the lower-dimensional space [8].
To recap, each in our dataset is vectorized in order to get an ensemble of vectors. We then apply TSNE clustering to reduce these vectors to 2 dimensions and we represent the remaining 2 dimensional vectors in a plot. Knowing a priori the labels of each patient that has said each word, it is possible to visually differentiate the vocabulary of healthy and unhealthy patients. The result is as follows:

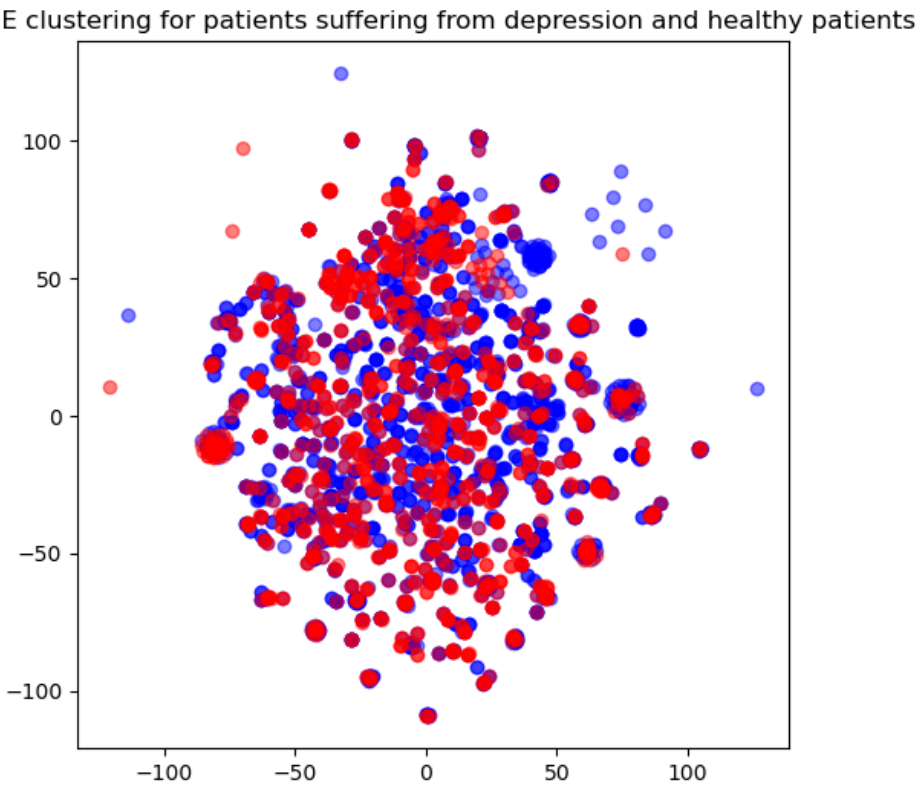


Fig. 12 dimensional representation of all the words used

Here, the plot shows all the vector representations of the words of healthy patients (blue) and unhealthy patients (red). Clearly, the semantic meaning of words used by both healthy and unhealthy patients are quite close in two dimensions. Most of the words here are shared and are overlapped. At first sight, classifying purely based on semantic meaning seems to be difficult. However, it is important to note that this is a projection in two dimensions from a 300 dimensional vector, meaning that a lot of data (that could be used for differentiating) is lost. This diagram is made to give a general idea on the complexity of the task.

2.3. Feature extraction

By vectorising the text that has been written out by patients, a model should be trained based on those vectors, and should be able to classify the depression risk of the patient (healthy / unhealthy). This means that these training vectors are very important. The goal of course is to get a model with the best accuracy. In order to get a baseline, let's first classify with standard ML models.

Despite having seen that classifying solely on word representation will be difficult. It is still possible to use it as a baseline. This baseline will allow for comparison to more advanced ML models that we will use later on.

As one type of input for these traditional ML models, the pooled average of the vector representation for each word will be used. That is to say, each patient will have one vector of fixed size 300 (from the spanish billion word corpus model) resulting from averaging the vector representation from each word that the patient has said.

Raw text	After Tokenization	After lemmatization	After vectorization	After pooling
"Today I am talking about classifying text data …"	"today", "I", "am", "talking", "about", "classifying", "text", "data"...	"today", "I", "be", "talk", "about", "classify", "text", "data"...	N Numerical vectors (0.2, 0.5, 0.54, 0.54, 0.43…..)	1 Numerical vector (0.1, 0.4, 0.4, 0.2…..)

In the end there is 1 vector (of size 300) assigned to 1 patient which has 1 label (0 or 1).

The second type of input that will be used is the word count which is a numerical representation of text data based on the frequency of words present in a given document. Just like before, text is tokenized, cleaned and lemmatized in order to normalize it.

Finally, TF IDF vectors will be used. Which, as described before, aims to quantify the importance of a word within a document or a corpus by considering both its frequency in the document and its rarity across the corpus. This technique also uses tokenization, cleaning and lemmatization in order to normalize our text.

2.4. Classification
2.4.1.Traditional ML (baseline)

These vectors will be our inputs to train our ML models. The results were recorded for a Logistic Regression, RandomForest and a Naive Bayes model. Hyperparameter optimization was done on all models and the best accuracy was kept. The results of the standard ML classifiers are in the ‘Results’ section of this document.

These first classification results are not bad and already allow us to achieve an 80% classification accuracy. This will be set as our baseline.

Limits of these models: Although these models are adapted for binary classification, it is the training input data that is quite poor. First of all, the vector representation of a word with word count does not capture its context in a sentence. Furthermore, by averaging all the vectors representing each word that a patient has said, a lot of information is lost (in the case of word embedding). These main issues make it difficult to get a better accuracy given the sample size.

All previous models were standard ML models. By using Deep Learning (DL) models, it is expected to achieve better results as the models might be able to pick up on hidden patterns in the data and specially have contextualized vectors providing more semantic information. There will also be a focus on different inputs as the previous types of inputs had major flaws.

2.4.2. Deep learning

2.4.2.1. Deep learning algorithms from scratch

Deep learning is a field of ML and artificial intelligence that focuses on training artificial neural networks to learn and complete certain tasks (classification, regression, generation). It is inspired by the structure and function of the human brain's neural networks.

DL algorithms are designed to automatically learn and extract hierarchical representations of data by analyzing large amounts of labeled examples. These algorithms are typically built using artificial neural networks, which consist of interconnected nodes (neurons) organized in layers. Each layer processes and transforms the data received from the previous layer, gradually learning more complex and abstract features as information propagates through the network [9].

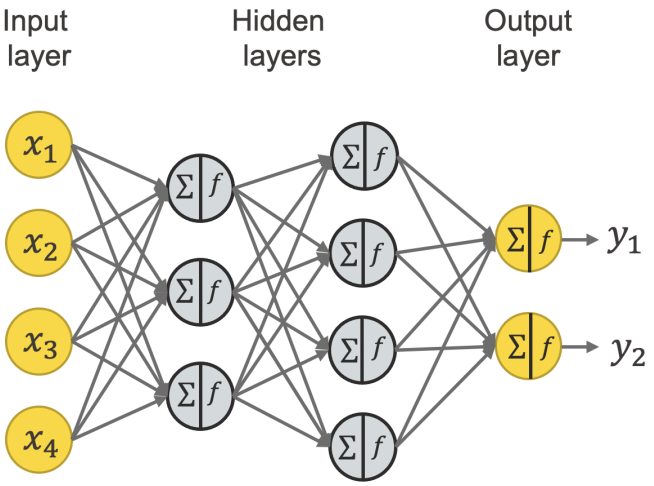


Fig 13. A neural network with a rather simple structure

In this diagram, each artificial neuron is represented by a node and each arrow represents the connection from the output of one artificial neuron to the input of another. Technically, the output of a neural network is no more than a function of its inputs.

Different architectures of neural networks will be more adapted to certain tasks. In our case, four different architectures were tested out: Deep Neural Network (DNN), Long Short Term Memory (LSTM), Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN).

Each model contains a large amount of weights and biases. It is important to remind ourselves that the training process of a DL model is finding out the best values for these weights and biases in order to produce the best accuracy possible.

In the case of this TFM, for the DNN, its parameters are the weights of each node. A DNN is a neural network with many layers of nodes and many nodes per layer. Traditionally, these nodes are ‘densely connected’, meaning that the input of one node is the sum of all the outputs from the nodes of the previous layer.

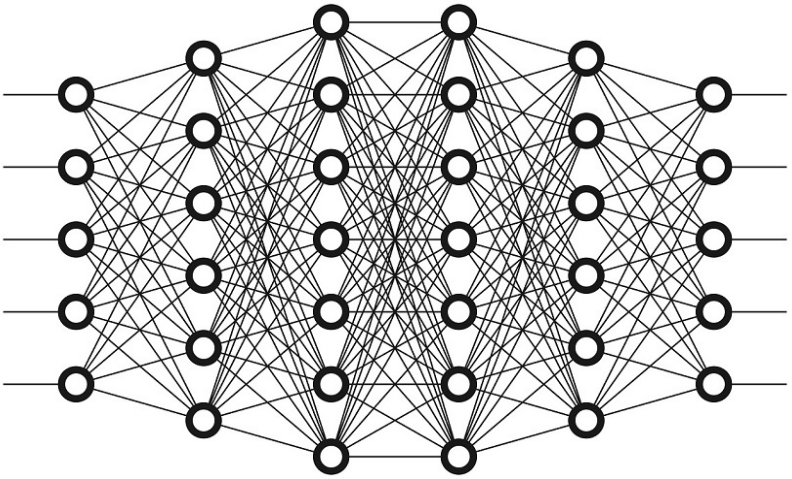


Fig 14. Example of a DNN architecture

For DL architectures, the number of parameters can quickly increase. Here, an optimal model was found with 7 layers and 120 000 parameters using Keras tuner. Furthermore, to help with the learning process, ‘dropout layers’ were added [10]. Dropout is a regularization technique often used in DNNs to prevent overfitting. Overfitting occurs when a model learns to perform well on the training data but fails to generalize to new, unseen data (validation data). Dropout helps address this issue by reducing the reliance of neurons on specific inputs and encourages the network to learn more robust and generalized representations.

In dropout, during the training phase, randomly selected neurons in a layer are temporarily "dropped out" or deactivated with a probability p. Due to the nature of our task, our probability was quite high (50%). When a neuron is dropped out, its outputs are ignored during that particular forward and backward pass. The selection of neurons to drop out is independent for each training example, and the dropped-out neurons change randomly from one training example to another.

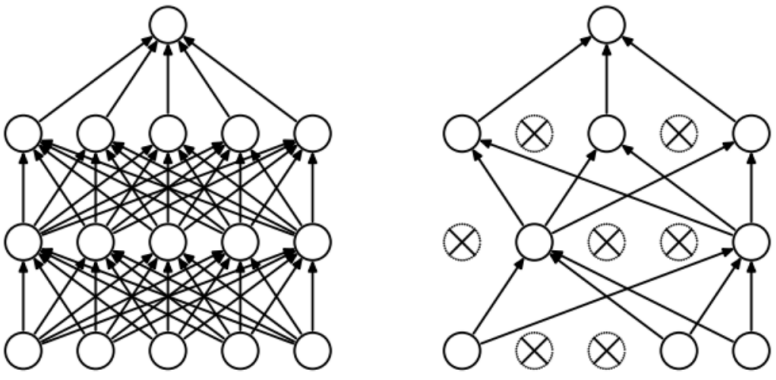


Fig 15. Visualizing dropout

Previously, Recurrent Neural Networks (RNNs) were used. RNN are a class of neural network architectures designed to process sequential data by maintaining hidden states that

capture information from previous inputs. RNNs have a recurrent connection, allowing them to retain and utilize information from previous time steps.

RNNs have a rather simple architecture, typically consisting of a single recurrent layer. However, they are often combined with other types of layers to form more complex architectures. In our case, dense neural layers were added to achieve better results.

RNNs are well-suited for tasks that involve sequential data, which is the case of our text data. They can effectively model dependencies between elements in a sequence and generate predictions based on the context of previous inputs [11].

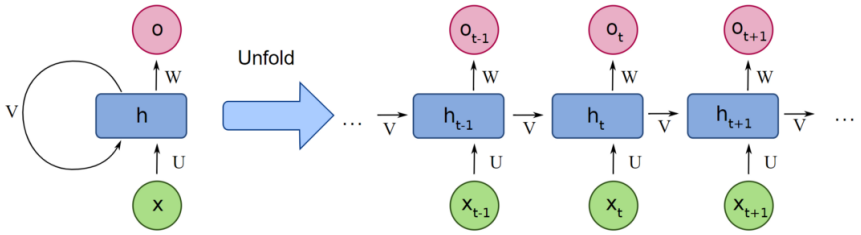


Fig 16. Illustration of the recurrent characteristic of RNNs

Despite them being theoretically adapted to our problem, RNNs suffer from the ‘vanishing gradient problem’, which hinders training, as it is possible to see in the results. The RNN model used had 324,160 parameters. Its architecture was as follows:

Layer Type	Output Shape	Number of parameters
embedding (Embedding) simple_rnn (SimpleRNN) simple_rnn_1 (SimpleRNN)	(None, None, 32) (None, None, 32) (None, 32)	320,000 2,080 2,080

Long Short-Term Memory (LSTM) is a type of RNN architecture that addresses the vanishing gradient problem and is widely used for processing sequential data.

LSTM networks have memory cells that allow them to remember information for long periods, making them capable of capturing long-term dependencies in sequential data. The architecture contains different cells that allow for efficient information management. Like the RNN, LSTMs are designed to process sequential data, but have been designed to avoid the vanishing gradient problem, which allows for much better training. LSTMs are generally larger models, in our case, the best performing model had 1,428,881 parameters. Its architecture was an embedding layer, followed by the LSTM, a dropout layer and a dense layer with 433 neurons. Despite its positive results (see ‘Results’ section), this total number of parameters seems too high for the task at hand and puts this model at a risk of overfitting.

Finally, Convolutional Neural Networks (CNNs) are another class of DL neural network architectures primarily designed for analyzing visual data, such as images. CNNs are adapted at capturing spatial patterns and hierarchical representations in data.

The core component of a CNN is the convolutional layer. Convolutional layers apply filters/kernels to input data, sliding them across the input spatially and computing element-wise multiplications and summations. This process allows the network to extract local features, such as edges, corners, and textures, and learn hierarchical representations through multiple convolutional layers.

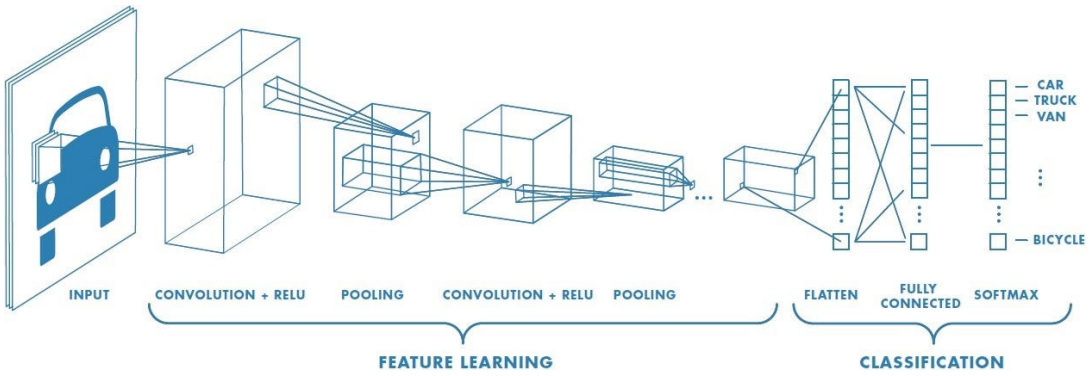


Fig 17. Visualization of a CNN

Although primarily used for image data and a priori not for text/ sequential data, the CNN architecture is able to help detect features and find patterns in the input data, meaning they have their place in text classification. In our case, a 1 dimensional CNN along with a LSTM with a specific architecture was used and was adjusted to the type of task at hand [12]. This model had 394,001 trainable parameters. Its architecture was as follows:

Layer Type	Output Shape	Number of parameters
embedding	(1, 32)	320,000
conv1d	(1, 40)	5,160
batch_normalization	(1, 40)	160
spatial_dropout1d	(1, 40)	0
max_pooling1d	(1, 40)	0
conv1d_1	(1, 40)	6,440
batch_normalization_1	(1, 40)	160
spatial_dropout1d_1	(1, 40)	0
max_pooling1d_1	(1, 40)	0
bidirectional	(1, 128)	53,760
dense	(1, 64)	8,256
dense_1	(1, 1)	65

As inputs for these models, an integer vectorizer will be used. This gives an integer vector for each “document”, or “text”. In this vector, each position corresponds to a unique word in the vocabulary. For each word in the sentence, the corresponding position in the vector is set to an integer value, representing the word.

In this case, the “document” being all the words that a patient has said. The document is vectorized and therefore gives out 1 vector per patient. This vector will be the training data. Furthermore, the first layer of each of these DL models is an embedding layer. This layer maps an embedding vector to each value of the input. The resulting vector is then an input for the following layers. It is important to note that this layer is trained too, meaning that the embedding vectors for all input change during the training process.

These different DL models were trained: Deep Neural Network (DNN), Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN) and a Bidirectional 1D Convolutional Neural Network (CNN). All models (except for the CNN, due to lack of computing resources) underwent hyperparameter search with the goal of getting the best validation accuracy. A hyperparameter is a parameter that is set before the learning process of a ML model begins. Unlike the model's internal parameters (weights and biases), which are learned through training, hyperparameters control the behavior of the learning algorithm itself. They impact the nature of the model itself, here are a few examples that were used in the hyperparameter search in this task: the number of layers in a DNN, the number of nodes in a layer, the dropout rate (probability of neurons to be dropped out in a layer) and the learning rate, which determines the step size at each iteration during the gradient-based optimization process. The exhaustive evaluation metric results of these models are in the ‘Results’ section of this document.

These models with these input representations do not always beat the regular ML algorithms, even though it was expected of them to. Even after extensive hyperparameter search that included flexible management of dropout layers, these more complex models fail to get consistent validation accuracy. Most of the time, their training/validation curves for loss and accuracy are very ‘unstable’. The sample size of our dataset is reasonable for ML models but too scarce for DL architectures, this is the reason why we deal with overfitting. Moreover, some DL models are made for more complex tasks (machine translation, text generation, etc.) than just simple text classification.

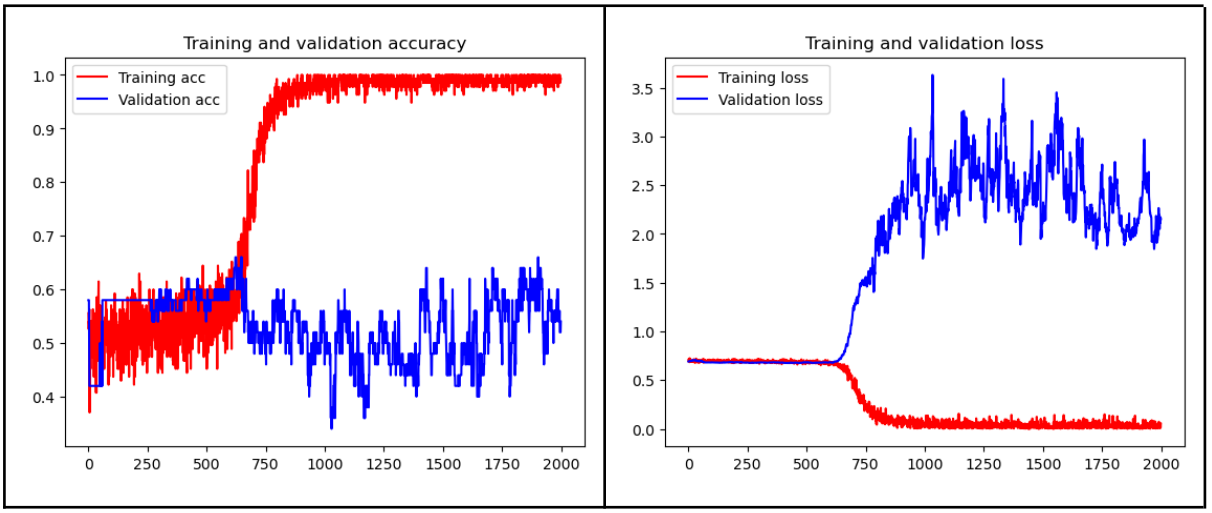


Fig 18. Training results for the CNN model, there is obvious overfitting

It is important to keep in mind that these results are for a CNN model that did, more or less, work. Without hyperparameter search, these models overfit completely.

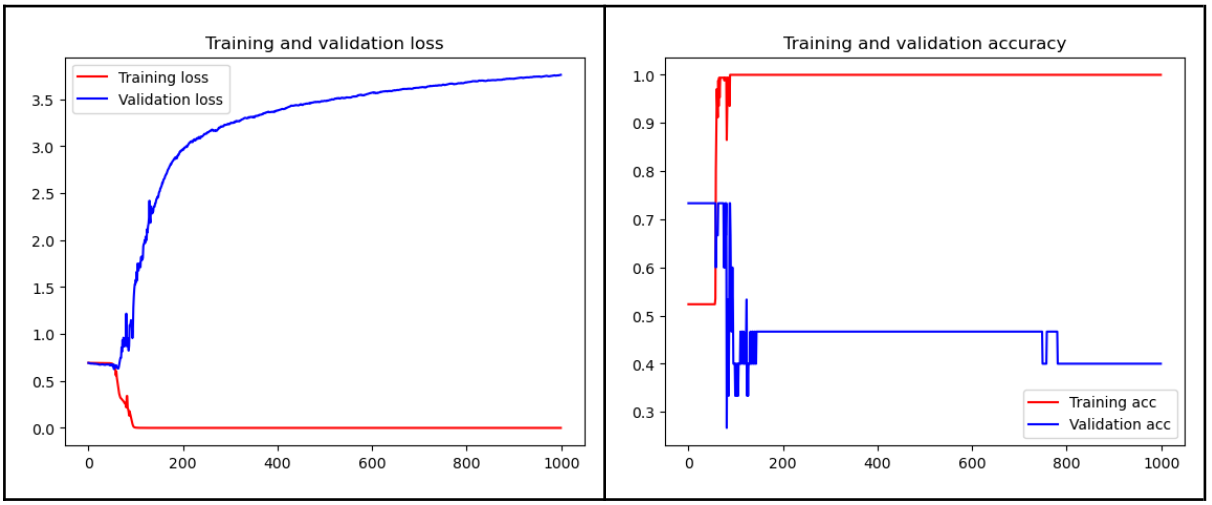


Fig 19. Overfitting in a regular LSTM model without hyperparameter adjustment

Some models like the RNN seem to not be adapted at all to this task. Suffering from gradient vanishing, the model fails to train or completely overfits and gives disappointing results. These models stagnate in accuracy scores despite being rather complex and despite large hyperparameter search. One reason that this might happen is that the input vectors fail to correctly represent our data. Even though representing each word using a dense vector using a pretrained model might seem useful at first, there is no easy way to combine all the vectors in order to have 1 representative vector per patient. To achieve a higher accuracy, it is necessary to not just look at the vocabulary that each patient uses, but also the way that it is used, and mostly try to encode the meaning of the written text so that classifying algorithms are able to use it. This calls for more powerful feature extraction processes.

2.4.2.2. Fine tuning BERT in Spanish for text classification

In order to solve these previously mentioned issues, a bigger, more powerful model will be used for encoding and embedding our text data. These types of models, BERTs, are common in the NLP landscape. BERT [13] (Bidirectional Encoder Representations from Transformers) is a popular pre-trained model developed by Google's AI team. It is designed to capture the contextual representations of words in a text using a transformer-based architecture. A transformer is a type of neural network architecture and is designed to capture relationships between different positions within a sequence. This attention mechanism allows the model to weigh the importance of different parts of the input sequence when making predictions.

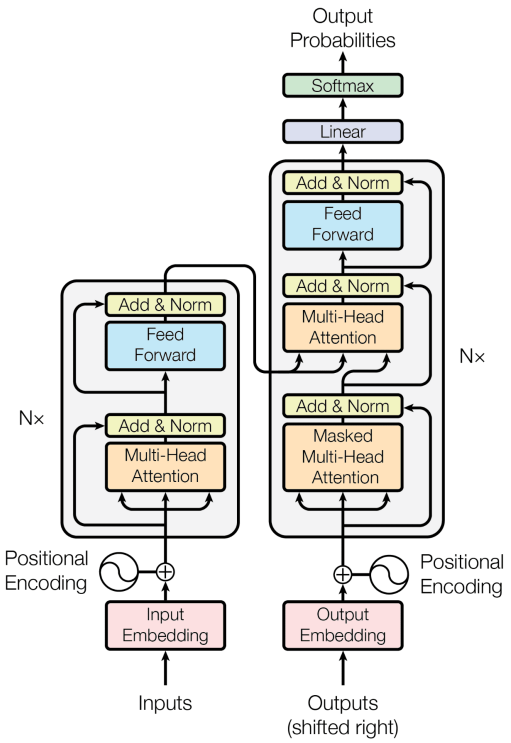


Fig 20. The transformer architecture from the original paper

According to the Google team responsible for the development of the transformer. The key components of a transformer model are: positional encoding, attention and self attention [14].
Positional encoding incorporates information about sequence order into the input embeddings of transformer models. Attention mechanisms allow the model to focus on relevant parts of the input sequence when making predictions. Self-attention, a specific type of attention, captures relationships between different positions in the sequence and helps model long-range dependencies effectively. These components are key to the success of transformer models in various NLP tasks.

More precisely, in our case, we will be using a BERT-Multilingual model since the text data is in Spanish.

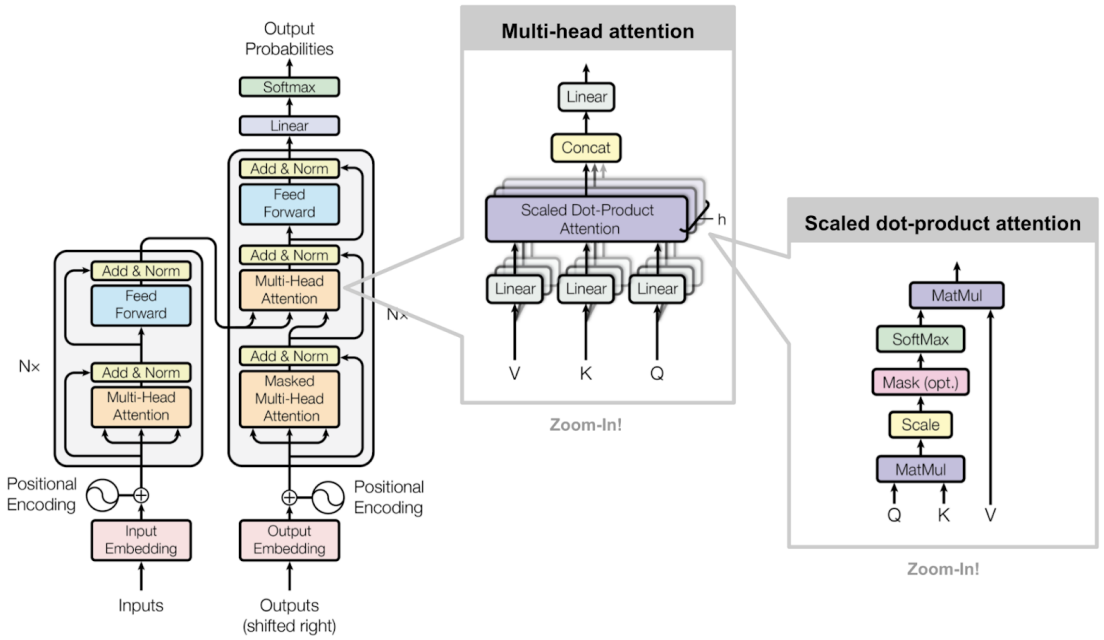


Fig 21. A more detailed description of the transformer architecture applied to BERT

The main reasons why one might want to use BERT models are:

- Transformer architecture: BERT is built on transformers, which are DL model architectures. Transformers are able to weigh the importance of different words in a sentence.
- Fine-tuning: BERT comes pre-trained and they can be further fine-tuned for specific tasks such as text classification (which is the objective in our case). During fine-tuning, the model is trained on labeled data with task-specific objectives, allowing it to adapt to the specifics of the target task.
- Deep Contextual Representations: BERT generates deep contextual word representations that capture the meaning and relationships between words based on the surrounding context. These representations are very useful for us in our case as inputs for classification models.

In this context, the BERT model is a text encoder that can be fine-tuned (adapted to the classification task) and that would feed into a classifying algorithm (a deep neural network). Since these BERT models are designed to capture meaning of words while taking into account the context that they are in, they should solve our previous issues that we've had regarding the vector representation of our data.

Like before, the encoder can give us one fixed-size vector representation per word. But, it can also output a fixed-sized vector per ensemble of text (like a sentence or group of sentences that a patient has written). Indeed, the BERT model is able to pool the N vectors representing a group of words as one vector that will be our input for our classification layer.

BERT utilizes a pooling strategy called "CLS pooling" to obtain a fixed-size representation for the entire input sequence. The CLS (CLaSsification) token represents the classification

aspect of BERT. During training, the model is fine-tuned to predict the task-specific output based on the representation of the CLS token (like text classification for example).
The pooling operation in BERT involves extracting and transforming the hidden state (vector) of the CLS token, rather than simply averaging the hidden states (vectors) of all tokens / words in the input sequence, BERT usually applies a non-linear transformation (like applying a dense layer with activation function) to the CLS token's hidden state to obtain a fixed-size pooled representation. What is important to remember is that this pooling captures the overall information or context of the input sequence that is relevant for classification or other tasks.

This hidden state carries information about the entire input sequence. Instead of simple averaging
The purpose of this pooled representation is to capture the overall meaning or context of the input sequence, which can be used as a summary or representation for downstream tasks like text classification in our case.

Using the fine-tunable BERT encoder and a dense neural network it is possible to have a complete classifier that should be able to capture the useful meaning of the text input in a way that is tailored to the task: binary text classification.

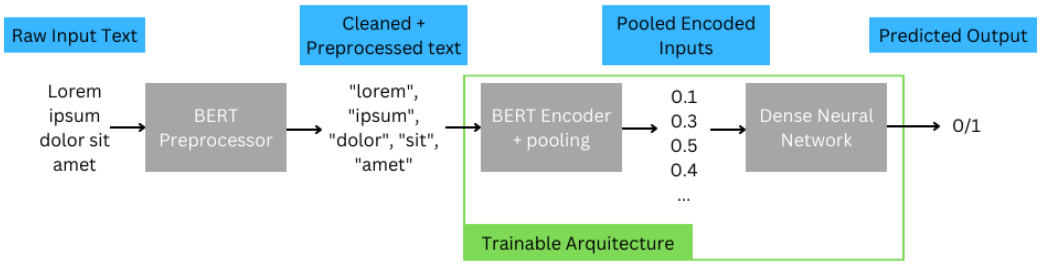


Fig 22. Summary of the classifying process

2.5. Data augmentation using text generative DL techniques

Because the main issue of our task is the lack of data, it would be helpful to generate new samples or text that would correspond to unhealthy or healthy individuals. This is far from trivial, in order to generate such tasks, we have to use a model that is already capable of generating spanish text and fine tune it to make it able to generate both “healthy text” and “unhealthy text”. This idea of generating new data samples is common in NLP and ML in general.

In this case, the “flax-community/gpt-2-spanish” [15] will be used, which is a Generative Pre-Trained model adapted to Spanish. It was developed by OpenAI and was designed to complete text tasks such as text generation, question answering and more. This model will

be run in Python. This GPT-2 model architecture consists of a stack of transformer layers. Each transformer layer consists of a self-attention mechanism and feed-forward neural networks. The model takes a sequence of input tokens and processes them in a parallel and hierarchical manner, capturing the dependencies and relationships between different tokens. This architecture allows the model to generate high-quality text by effectively modeling long-range dependencies and contextual information.

The generative model is fine tuned by the use of a ‘control code’. To generate text a healthy patient would say, it is possible to modify the tokenizer by adding special tokens to handle the control code. This code is just the name that we want to give to our task (generating ‘healthy’ text). If it is named ‘healthy’ and the tokenizer is modified by adding special tokens to handle the control code, it is possible to guide the model's generation towards a specific theme or topic related to "healthy" text. By adding this control code as a special token, it is possible to prompt the model to generate text that aligns with the "healthy" theme. This control code will be joined with our original ‘healthy’ text data to compose our training data used for fine-tuning.

Regarding the loss function used in the training of this model, it employs a language modeling loss. Language modeling loss is commonly used in generative models like GPT-2 to optimize the model's parameters during training. The loss function measures the discrepancy between the predicted probability distribution over the next token and the true next token in the training data [16].

As for the evaluation metrics, common metrics used for evaluating generative models like GPT-2 include perplexity and human evaluation. The original GPT-2 model was trained with human evaluation and perplexity but it is not possible to use it for the fine-tuning part of this project for lack of resources. Perplexity measures how well the model predicts the next token given the context, with lower perplexity indicating better performance.

Once the model is trained (fine-tuned), it is possible to generate an arbitrary amount of ‘healthy’ text. So, around 5 times the number of original samples was generated. This process is repeated for “unhealthy” text (starting from a new fresh model), and generates around 5 times the amount of original samples as before. This would increase the total sample count from 185 to 1055. With this entire new quantity of new data, it is possible to greatly expand the training dataset of the classification model.

In theory this works to increase the amount of sample in the training set however. In the case of this TFM, it was difficult to produce any results locally despite having the working code as these models generally need large amounts of computing power to train. A rough estimate to fine tune the generative model was around 150 hours of training time on the local computer.

Locally, it took 2 hours to achieve 3% of an epoch. The model would need at least 1 or 2 epochs to get usable generated results. This is where Google Collab was used as it allowed access to powerful computing resources like the NVIDIA T4 Tensor Core GPU which far outperforms the CPU used locally. The training process took an hour in total for both ‘healthy’ and ‘unhealthy’ data and new data was subsequently generated.

The generated text was exhaustively analyzed, and the most realistic text was selected by comparing it to the original data. With this new data, the dataset size is greatly increased. With this new data, the standard ML models as well as the advanced BERT classifying model can be trained again and should achieve better accuracies.

Here is an example of generated text for a healthy patient: “yo me siento feliz con la gente, con mi familia” and this is an example text from an ‘unhealthy’ patient “en la noche me dormí y me desperté, me sentí mal, pero bueno me levanté, no sé cómo dormir”.

With this new data, it is possible to train the standard ML models once again. The more advanced BERT model has also been re-trained. It is expected for these classification models to achieve better validation accuracies in general when they have access to more training and validation data.

3. Results

This section will present the results from all the experiments explained in the methodology, taking special attention to the accuracy as the evaluation metric even though confusion matrix and ROC curves where also used to choose the best model:

For traditional ML models:

	Model			
Feature extraction (with original dataset)		Logistic Regression	Random Forest	Naive Bayes
	Word count	75%	70%	55%
	TFIDF	71%	71%	68%
	Word Embedding	80%	77%	65%
Feature extraction (with data augmentation)	Word Count with data augmentation	85%	**88%**	58%
	TFIDF with data augmentation	87%	85%	82%
	Word Embedding with data augmentation	82%	80%	72%

Fig 23. Machine Learning models and their corresponding validation accuracies

For this binary classification task, ML models perform quite well achieving 70%-80% accuracies with the original data. Furthermore, data augmentation adds 10 to 15 percentage points depending on the model with the best result being 88% validation accuracy. This is a large improvement. This 88% accuracy is an average of a 5 fold cross-validation accuracy. Overall it seems that random forest and logistic regressions are the most adapted for this binary classification task.

Deep Learning Models:

	DNN	LSTM	RNN	CNN	BERT fine tuned
Original data	75%	**85%**	56%	64%	75%
Data augmentation	-	-	-	-	**87%**

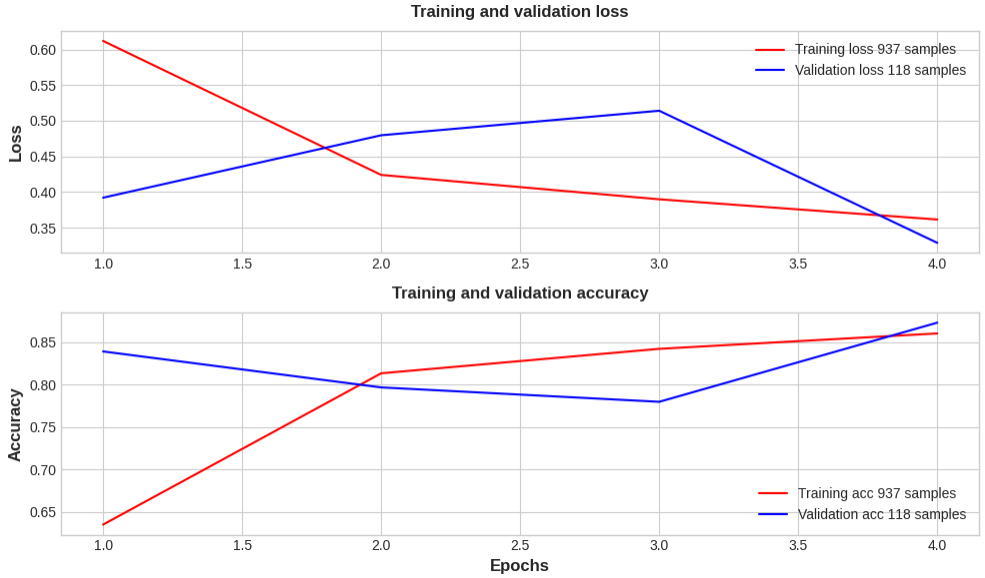


Fig 24. BERT Model training process

Out of these five models, two stand out because of their results. The LSTM model performs well with the original 185 samples (85% accuracy). But the fine-tuned BERT model along with the generated samples achieves even better, 87%.

Even though better results were expected with DL, clearly it sometimes fails to even surpass standard ML accuracies. Looking at the DL training curves, the accuracy curves are very jagged. This is likely due to the fact that the 185 samples are split into 165 training / 20 evaluations. The evaluation accuracies can only take a few samples. Thus, our hypothesis is

that with more data or further regularization on the training process (L1 penalization, L2 penalization, dropout layers, dropout values), the validation accuracy would improve. However, the hyperparameter search of the neural network was quite flexible, meaning that the Keras tuner was able to look for the optimal amount of dropout layers to use and the optimal dropout probability value. This means that any more regularization would probably have resulted in stagnant losses and general incapacity of the model to learn. Despite using a powerful model such as BERT, fine-tuning it and achieving reasonable accuracies, there is not enough original data to achieve better results. Moreover, due to computation power constraints, it was not possible to look for hyperparameters while fine-tuning the BERT model. An accuracy that surpasses 88% could have been achieved since this is a priori the best way to classify text data.

Having analyzed the capabilities of the different classification models, we state that the limited quantity of sample, the models overall perform quite well. Indeed, using only 185 patients, these models are able to achieve 80% accuracy, even for the basic ML models (the best was 88% accuracy for the Random Forest classifier).
For the DL models, due to computational limits, the time needed to get these models running was very long and slowed down a lot of the work. It could have been possible to optimize the architecture of the BERT model to get more stable training curves, more accurate penalization and to finally get a higher final validation accuracy. Probably using samples and performing hyperparameter search on these pre-trained models, their corresponding accuracies would have likely increased up to 90% or even more. According to the literature, this BERT is a priori the best performing model for NLP.

Summarizing the results, standard ML achieves 88% being the best accuracy. Using the state of the art BERT model and data augmentation techniques, 87% validation accuracy is achieved. Achieving very similar results with a very complex and heavy model might not be worth the computational cost and effort for both training and running. Standard ML seems to work here because the problem is binary classification and it is a relatively simple task similar to sentiment analysis. It means that these ML models are good baselines. On the other hand, DL models are generally best suited for complex tasks (text generation, translation etc…). Indeed, we intuitively know that the more advanced models using BERT could perform better as they are a standard for NLP tasks. However, with the limited resources for this project, in regards to the quantity of data and computational resources, it is difficult to fully use/train these models to get solid results for this study.. As next steps, larger databases are required or more generative DL should be used for data augmentation in order to get better classification accuracies.

Although it was possible to find a solution to increase the accuracy of the algorithm and reduce overfitting, several limitations have been found during the process. The first one is the **lack of data.** This is the most common issue in these unstructured data tasks. A rule of thumb is to have ~1000 samples per class. This project only had about 100 samples per class. Because this is a common issue, data augmentation techniques like text generation are commonly used too in order to help with the training process. In terms of documentation and methodology, this project was also limited due to its nature of dealing with **Spanish text**. The model itself is of course limited to depression detection for Spanish speaking users. Furthermore, most open source and freely available documentation and code is made for the

English language. If English was the original data, the final accuracy could have been better. This mostly applies for the English BERT that would have been used in the encoding process, which is the current state of the art model.

Finally, the biggest issue was a **lack of computational resources.** This would have not only made all the hyperparameter search, optimization and data handling a lot quicker, but would have allowed for more thorough hyperparameter search in certain models. It could also have allowed for hyperparameter optimization where previously it was impossible. For the BERT model for example, it was not possible to do a hyperparameter search because of the size of the model. It is not possible to know how the model's performances would have increased if such a search would have been done, but probably by a few percentage points. This would have made the difference and gotten the model a lot closer to a 90 or even 95% validation accuracy.

4. Conclusion

Even though some limitations due to data restrictions were found, encouraging results classifying depressive patients have been achieved. Indeed, after having completed an extensive search on the best way to encode the data and to classify it, the best achieving model showed an impressive 88% percent accuracy with the reduced data sample available. It is certain that with further computation and/or even more data, the BERT model (and probably also ML models) could surpass the 90% accuracy threshold thanks to its powerful architecture.

This data processing pipeline could also not only be used for depression classification, but it also has the potential to be applied to other mental illnesses such as eating disorders, schizophrenia, addicting behaviors, etc. With more data and more ressources, even a multiclass model could certainly be achieving accuracies superior to 90%.

Taking a look at the original data, patients were talking about their illness and their day-to-day feelings. The topic of conversation was already centered around mental health. For such a model to scale and generalize, it would be interesting to not only analyze discussion about mental health, but also about everyday conversations, mundane topics, as it seems that segregating information could also be extracted by how an individual talks in other contexts.

Furthermore, this model is trained on a relatively small amount of data, if such a model was to be deployed at scale (at the scale of the Telegram application), it should be trained on a much larger volume of data, using more varied data. Moreover, the text data we used since the very beginning was mainly based on people talking about their own experience with mental health issues. Firstly, depression risks could be detected when people talk about other topics, or more generally, in their everyday vocabulary. This is to say that it might be interesting to use different conversations/ text messages to vary the data. Of course this would have to be done in a strict, privacy-respecting way, in order to follow RGPD guidelines. Secondly, other types of inputs could be useful for the classification. For text

messages, this could be the time at which the messages are sent, the frequency of message sending or some other metadata linked to the text message.

It is also interesting to note that even on the scale of a project with very few resources, a model was able to be built with an accuracy close to 90%. This gives an idea as to what could be possible at the level of a company or even at the scale of a data oriented company like Telegram or Facebook/Meta. Along with their ability to reach millions of people, if such companies were able to implement powerful models all the while respecting the privacy of their users, the total impact could be substantial.

Overall, AI seems quite a promising tool supporting clinicians on early detection of depression using automatic NLP techniques having the potential to make a huge impact, making people happier and even saving many lives worldwide.

References

[1] “The Importance of Early Detection and Diagnosis of Mental Health Conditions, Dementia and Learning Disabilities.” *Active Social Care*, https://activesocialcare.com/handbook/mental-health-dementia-and-learning-disabilities/the-importance-of-early-detection-and-diagnosis-of-mental-health-conditions-dementia-and-learning-disabilities/. Accessed 15 June 2023.

[2] Riggio, Christopher. “What's the deal with Accuracy, Precision, Recall and F1?” *Towards Data Science*, https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021. Accessed 15 June 2023.

[3] *YouTube*, 9 May 2023, https://sites.google.com/view/mentalriskes/home?authuser=0. Accessed 15 June 2023.

[4] Ahmadi, Fardin. “Large-Scale Textual Datasets and Deep Learning for the Prediction of Depressed Symptoms.” *NCBI*, 12 April 2022, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9019419/. Accessed 15 June 2023.

[5] MentalRiskES, https://sites.google.com/view/mentalriskes/home?authuser=0. Accessed 15 June 2023.

[6] *TFIDF statistics | SAX-VSM*, 27 October 2016, https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html. Accessed 15 June 2023.

[7] “Spanish Billion Word Corpus and Embeddings.” *Cristian Cardellino*, https://crscardellino.ar/SBWCE/. Accessed 15 June 2023.

[8] “t-distributed stochastic neighbor embedding.” *Wikipedia*, https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding. Accessed 15 June 2023.

[9] Melcher, Kathrin. “A Friendly Introduction to [Deep] Neural Networks.” *KNIME*, 23 August 2021, https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks. Accessed 15 June 2023.

[10] Brownlee, Jason. “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks - MachineLearningMastery.com.” *Machine Learning Mastery*, 3 December 2018, https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. Accessed 15 June 2023.

[11] Kosko, Bart. “Recurrent neural network.” *Wikipedia*, https://en.wikipedia.org/wiki/Recurrent_neural_network. Accessed 15 June 2023.

[12] MVG, Jose. https://github.com/Josemvg/humor-hound/tree/main. Accessed 15 June 2023.

[13] “[1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *arXiv*, 11 October 2018, https://arxiv.org/abs/1810.04805v2. Accessed 15 June 2023.

[14] “Transformers, explained: Understand the model behind GPT, BERT, and T5.” *YouTube*, 18 August 2021, https://www.youtube.com/watch?v=SZorAJ4I-sA&ab_channel=GoogleCloudTech. Accessed 15 June 2023.

[15] “flax-community/gpt-2-spanish · Hugging Face.” *Hugging Face*, 1 June 2023, https://huggingface.co/flax-community/gpt-2-spanish. Accessed 15 June 2023.

[16] “Language Models are Unsupervised Multitask Learners.” *2018*, https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf. Accessed 15 June 2023.

[17] “Introduction to Balanced and Imbalanced Datasets in Machine Learning.” *Encord*, 11 November 2022, https://encord.com/blog/an-introduction-to-balanced-and-imbalanced-datasets-in-machine-learning/. Accessed 15 June 2023.