UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DEVELOPMENT OF A MACHINE LEARNING SYSTEM FOR PREDICTING STRESS AT THE WORKPLACE

PABLO SAINZ SAN JUAN
ENERO 2022

TRABAJO DE FIN DE GRADO

Título:	Desarrollo de un Sistema de Aprendizaje Automático para Predecir el Estrés en el Centro de Trabajo.
Título (inglés):	Development of a Machine Learning System for Predicting Stress at the Workplace
Autor:	PABLO SAINZ SAN JUAN
Tutor:	SERGIO MUÑOZ LÓPEZ
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DEVELOPMENT OF A MACHINE LEARNING SYSTEM FOR PREDICTING STRESS AT THE WORKPLACE

PABLO SAINZ SAN JUAN

ENERO 2022

Resumen

Hoy en día el estrés es un fenómeno muy extendido que aparece en nuestras vidas de forma consciente o inconsciente, siendo el causante de una gran proporción de las bajas laborales. Aunque un cierto nivel de estrés podría actuar como un factor positivo, su exposición prolongada deriva en efectos perjudiciales para la salud y puede desencadenar o empeorar enfermedades como depresión, ataques de pánico, hipertensión, diabetes y problemas cardíacos.

Gracias al Aprendizaje Automático es posible predecir el nivel de estrés de forma no invasiva mediante técnicas como el análisis del sentimiento presente en el texto, el reconocimiento de la postura corporal, las variaciones en la presión cardiaca o incluso midiendo la conductancia de la piel. Estas últimas implican un fuerte desemboloso inicial ya que son necesarios sensores capaces de detectar con precisión dichas variaciones fisiologicas. En este proyecto se ha apostado por técnicas de detección que eviten la necesidad de adquirir hardware específico para su uso pudiendo encontrar todos los elementos necesarios en un ordenador.

El objetivo será, por tanto, el desarrollo de un sistema de detección de estrés a partir del uso del ordenador. Para ello, se utilizarán técnicas de aprendizaje automático y técnicas de Procesamiento de Lenguaje Natural (PLN) que permitirá desarrollar dos modelos: uno capaz de detectar el estrés presente dentro de un corpus de texto, y otro en función del uso que se hace del teclado. Estos modelos, se desarrollarán y se validarán por separado, para posteriormente alojarlos en una aplicación propia que sea capaz de predecir el estrés de un usuario.

Para la creación de dichos modelos se empleará el lenguaje de progamación Python y para el desarrollo de la aplicación se empleará la herramienta Flutter.

Palabras clave: Estrés, Aprendizaje Automático, sentimientos, PNL, Python, Flutter, texto, teclado

Abstract

Nowadays stress is a widespread phenomenon that appears in our lives consciously or unconsciously, causing a high percentage of sick leaves at work. Although a certain level of stress could act as a positive factor, prolonged exposure to stress leads to detrimental health effects and can trigger or worsen diseases such as depression, panic attacks, hypertension, diabetes and heart problems.

Thanks to Machine Learning it is possible to predict the level of stress non-invasively using techniques such as body posture recognition, analyzing variations in heart pressure, pulse or even measuring skin conductance. These techniques involve a high initial outlay since they require sensors capable of accurately detecting these physiological variations. For this project we have chosen techniques such as text sentiment analysis or the keystroke dynamics analysis, since they do not require the purchase of specific hardware and all the necessary elements can be found in a computer.

Therefore, the objective will be the development of a stress detection system. In order to achieve this, we will use Machine Learning and Natural Language Processing (NLP) techniques that will allow us to develop two models: one capable of detecting stress within a corpus of text, and the other based on the use made of the keyboard. Both models will be developed and validated separately, to be subsequently housed in a self-designed application capable of predicting a user's stress.

Python programming language will be used for the creation of these models and Flutter will be used for developing the application.

Keywords: Stress, Machine Learning, sentiment, NLP, Python, Flutter, text, keystroke dynamics

Agradecimientos

Me gustaría dar las gracias a mis padres por su incesante apoyo durante esta etapa universitaria. Sin vosotros hubiese sido imposible llegar hasta aqui.

No me olvido de todos esos amigos que me han acompañado en este viaje, desde dentro o desde fuera de la universidad. Ni de Raúl y Víctor que desde un primer momento me hicieron perder el miedo a esta carrera. Gracias a todos, sois mi segunda familia.

Finalmente agradecer al GSI por haberme dado la oportunidad de realizar este trabajo y, en particular, a mi tutor Sergio por su apoyo y su confianza desde el primer minuto.

Contents

List of Figures

Tables Index

CHAPTER 1

Introduction

1.1 Context

Nowadays, stress is a word frequently used to denote a disease that comes as a result of routine pressures and the several adversities that have negative impacts on health; caused by the frenetic lifestyle of modern society. It is a concerning disease that does not make any distinction of age, gender, economic condition or race; and it is on the rise worldwide. Nonetheless, it is essential to know what the meaning of the word "stress" really is.

Stress is a vast concept that refers to psychological and biological processes that occur due to challenging emotional and cognitive situations [1]. It can be considered as a state of physical and psychological tension in response to a stimulus or pressure, either a positive or a negative one. At the same time it is a defense mechanism that, in small proportions, helps the organism to respond to the stimuli of daily life and adapts it to the upcoming events. However, stress usually appears out of control and can become a chronic condition, causing anxiety, psychosomatic disorders and worsening mental disorders such as depression.

When its occurrence is related to a working environment, then we speak of work stress. Work stress occurs when there is a mismatch between work demands and the resources available to the worker to cope with them. This causes a state of physical and mental

exhaustion that drives the employee to be less motivated, less productive and less confident at work. This tends to make the organization less successful in a competitive market. According to several estimates [2], job stress takes a considerable sum from the national economy in sick pay, productivity loss, health care and legal costs. These personal and economic costs have prompted the study and development of stress detection techniques.

At first, stress could be detected and quantified by administering questionnaires [3] or by medical procedures such as EGG [4]. But, with the emergence of the internet of things (IoT) and machine learning (ML), new stress detection methods and techniques emerged that transformed this field of research.

IoT enables better accuracy, real-time tracking, alerting, monitoring and reporting irrespective of any place or any time. IoT in healthcare provides customized personal service to the person under stress by utilizing individual gadgets and frameworks that are even utilized by single person. Simultaneously, new learning models capable of detecting stress based on biological signals [5] (such as variations in skin conductance, pulse analysis and heart pressure) are appearing. However, these techniques often require very specific hardware which increases costs. In recent years, research fields such as natural language processing (NLP) had emerge, bringing with them techniques capable of recognizing sentiment within text [6].

This work attempts to analyze the stress of a person by using his or her working tool, the computer [7]. The developed system will be able to analyze stress in a unobtrusive way by examining the sentiment of the typed text and analyzing the user's typing patterns [8]. This result in a decrease of material costs and in an easier deploy in work or academic environments.

We will study different classification techniques and compare them in order to establish a model capable of accurately detecting the stress of a person. For the training of these classification models, different datasets will be used depending on the model to be trained. Facing stress prediction from text, an existing dataset resulting from the analysis of the sentiment present within different posts in the Reddit social network will be used [9] along with another dataset made up of labeled tweets based on their hashtag content [10]. On the other hand for the analysis of writing patterns, we will use the dataset that was generated for the SWELL [11] project. Finally, these models will be embedded in a Flutter application so it can be deployed either in a web environment or in iOS and Android devices.

1.2 Project goals

The objectives of the project are the following.

- Conduction of a preliminary study of the stress detection techniques to be used in the project. This is intended not only to acquire the necessary data sets, but also to define the different classification models that will handle them.

- Prior cleaning of the datasets to avoiding the presence of elements that may affect the subsequent execution of the training models. In addition to cleaning, the labels of the datasets are going to be checked to make them balanced in case they are not calibrated.

- Utilization of different Machine Learning approaches for each set in order to draw conclusions from the data obtained. This will help to select and validate a stress detection model based on the available features.

- Development of an application using the Flutter SDK that embodies the above models and accurately detects user stress.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1. The study will be place in context and the objectives of the project will be presented. Also, a brief summary of how the information is going to be managed will be provided.

Chapter 2. Several technologies that have helped to bring this project through to completion,will be presented as well as some information on what they are for, where they come from and, briefly, how they will be used.

Chapter 3. Describes the design of a machine learning model for stress text detection, as well as its evaluation through experiments and the analysis of its results.

Chapter 4. Details the implementation of a classification model for stress detection based on a user's typing patterns. It is also presented its evaluation as well as the results obtained and the subsequent comparisons and deductions drawn from them.

Chapter 5. This chapter will describe the design and development of a Flutter application where the previously defined models will be integrated. The main components of the application and its workflow will also be defined.

Chapter 6. Overall conclusions obtained from this project, the objectives achieved, the problems encountered and their respective solutions, as well as suggestions for future work are presented.

CHAPTER 2

State of Art

2.1 Stress detection techniques

Nowadays, one of the great dilemmas of companies is the ability to maintain a stable level of stress among their workers. Since work stress has a significant impact on productivity, keeping stress at an adequate level means an increase in performance for the company. Maintaining that stability is a very sensitive process since a high level of stress can lead to anxiety or sick leave, and a low level can lead to a lack of motivation. However, it is very complex to recognize the stressors and therefore it is very difficult to regulate stress levels.

Stress at work has had a huge impact on companies, which has increased interest and research on stress prevention. Preventing work stress can be achieved by identifying the main stressors. Such a strategy could allow the design and implementation of regulatory policies to reduce the adverse effects of stress. However, evaluating the effectiveness of these policies in a real scenario is a challenge, mainly because of the costs involved in applying stress detection and regulation techniques in a realistic environment.

The objective of this project will therefore be to develop a stress detection method that can be deployed in work or academic environments. Therefore, in order to define the technologies to be used in our project, a prior analysis of the technologies applied in other

similar works was carried out.

Some studies[6] use as indicators of stress: behavioral manifestation, physiological signal, physical appearance, emotional manifestation, facial expression or voice recognition. Nowadays, analyze this stressors in a work environment is arduous since it requires an infrastructure that hosts sensors to enable the detection. Therefore, we will focus the study on non-invasive detection techniques that use the computer as a source of information.

Firstly, we focused our analysis on the different techniques used for stress recognition based on keystroke dynamics. During this process, we have found some studies that were capable of predicting stress by analyzing variations in typing by using an Android application[12]. Other projects developed a multi agent system (MAS)[13] which used sentiment and stress analyzers that processed text data, and sentiment and stress analyzers for keystroke dynamics data in an attempt to early prevent future issues caused by social media interactions.

SWELL project and Lisa M. Vizer's work[14] proposed various methods of stress detection using only the computer as the detection tool. For this purpose, they trained several classifiers with keystroke features achieving really good performance from them. This method of stress detection requires no additional hardware, is non-intrusive and is easily adaptable to individual users, which demonstrates its great potential.

Finally, Elsbeth Turcan's work, [9] and Genta Indra Winata's work [10] , expose different technologies and approaches in order to detect stress in text. They also provide two labeled datasets that will be used to train our supervised models.

Dreaddit is a new text corpus of lengthy multi-domain social media data for the identification of stress obtained from [9]. Consisting of 190K posts from five different categories of Reddit communities and label 3.5K total segments that where taken from 3K posts using Amazon Mechanical Turk. The study presented preliminary supervised learning methods for identifying stress, both neural and traditional, and the analysis of the complexity and diversity of the data and characteristics of each category.

Genta work, proposed a Long Short-Term Memory (LSTM) with an attention mechanism that classify psychological stress from self-conducted interview transcriptions. They applied distant supervision in order to automate tweet labeling based on their hashtag content, complementing and expanding, thus, the size of our corpus. This additional data was used to initialize the model parameters, improving the model's robustness, especially by expanding the vocabulary size. The bidirectional LSTM model with attention proved to be the best model in terms of accuracy (74.1%) and f-score (74.3%). In addition, adjusting for

distant supervision was found to improve model performance by 1.6% accuracy and 2.1% f-score. The attention mechanism helped the model to select informative words.

Once the related work in the field of stress detection has been analyzed, we proceed to describe the main technologies which are relevant of the implementation of the project.

2.2 Enabling technologies

2.2.1 Machine Learning Technologies

There are many definitions for Machine learning, but a general one [15] would describe it as a scientific discipline in the field of Artificial Intelligence that creates systems that learn automatically. In this context, learning means identifying complex patterns existing in huge quantities of data. Actually, the machine that learns is an algorithm that reviews the data and is able to predict future behaviors. In this context, automatic also means that these systems improve autonomously over time, without human intervention. There are 3 types of learning within machine learning: supervised learning, unsupervised learning, and reinforced learning.

- **Supervised learning** aims to predict a value (or label) by training a model on a set of previously labeled data. Therefore, the goal of supervised learning is to create a function capable of predicting the corresponding value for any valid input object after having seen a set of examples ("training data"). This requires the generalization from the presented data to situations not previously seen.

  A classification problem occurs when discrete values have to be predicted, for which techniques such as KNN or SVM are used. On the other hand, if continuous values are to be predicted, it is a regression problem and Linear or Logistic Regression algorithms will be used. This is achieved through two phases, training and testing. To do this, in our approach, the dataset will be distributed into two groups, generally allocating 75% of the data to training and the remaining 25% to testing.

- **Unsupervised Learning** tries to obtain patterns from unlabeled data. This type of learning applies a clustering to the data in order to separate it. After the clustering, data is divided in groups with similar characteristics. K-means is the most common algorithm for this type of learning.

- **Reinforcement Learning** is concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative rewards.

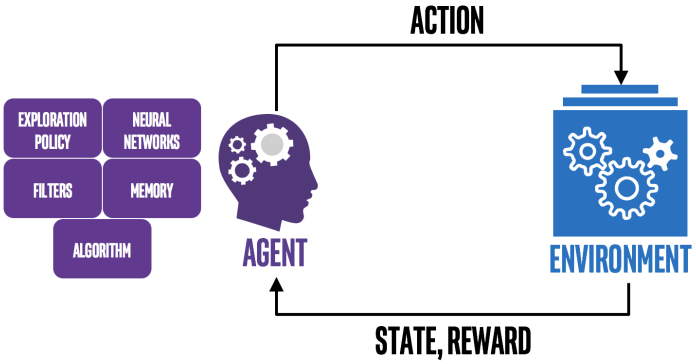

Figure 2.1: Reinforcement Learning schema

Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

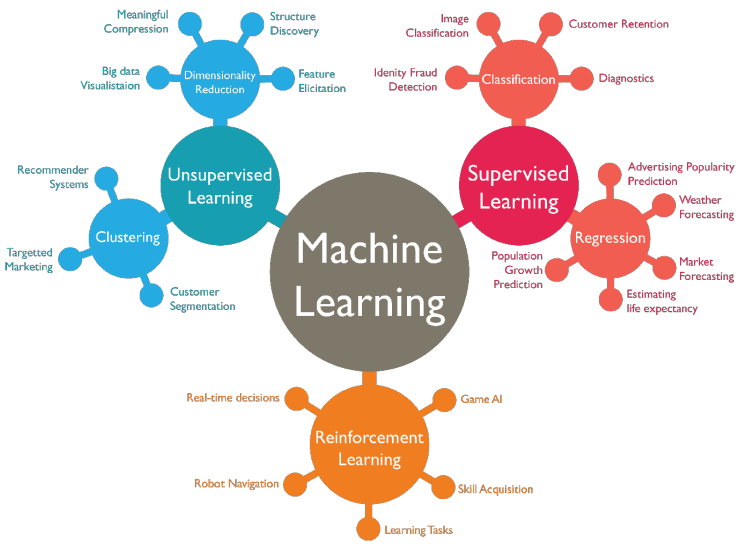

Figure 2.2: Types of Machine Learning

In this project supervised learning will be used due to the nature of the datasets used.

2.2.2 Python

Python is an interpreted programming language widely used in Machine learning and Big Data. As an interpreted language, the code is translated and executed simultaneously; in addition, variables could assume different values according to the code.

2.2.3 Python extensions used for Machine Learning

First, we present the extensions used for the processing and graphical representation of the information. These tools will help us to manage and clean the datasets as well as to visually represent their content. Some of the extensions used are:

Pandas will help us to handle the information during its analysis. It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Wes McKinney (main Developer of pandas) started working on it in 2008 due to a need he had for a flexible, high-performance tool to perform quantitative analysis on financial data while he was working for AQR. The main use of the tool in this project has been the manipulation of the datasets.

Numpy is a library for Python programming language that supports the creation of large multidimensional vectors and matrixes, along with a large collection of high-level mathematical functions to operate on them.

Matplotlib is a library for generating graphics from data stored in lists or arrays using Python programming language and its mathematical extension NumPy. It provides an API, Pylab, designed to be reminiscent of MATLAB.

Natural Language Toolkit The Natural Language Toolkit, or more commonly NLTK, is a set of libraries and programs for symbolic and statistical natural language processing (NLP) for the Python programming language which includes graphical demonstrations and sampled data. It provides more than 50 corpus and lexical resources that allows the classifying, tokenization and stemming of data between others.

We present below the different extensions that will allow us to develop the classification models:

Gensim is an open source library for unsupervised themed modeling and natural language processing using modern statistical machine learning. It is implemented in Python to improve performance. Gensim is designed to handle large text collections using data streams and online incremental algorithms, which makes it different from most other ma-

chine learning software packages that target only memory based processing.

Keras is an Open Source Neural Network library written in Python. It is capable of being executed on top of TensorFlow, Microsoft Cognitive Toolkit or Theano. It is particularly designed to allow experimentation in shorter time with Deep Learning networks. Its strengths are centered on being user-friendly, modular and extensible.

TensorFlow is an open source library for machine learning across a range of tasks, and developed by Google to satisfy its needs for systems that are capable of building and training neural networks in order to detect and decipher patterns and correlations, which are analogous to the thinking and learning that humans do.

Scikit-learn is a free software machine learning library for the Python programming language that includes several classification, regression and cluster analysis algorithms including support vector machines, random forests, Gradient boosting, K-means and DBSCAN. It is designed to interoperate with the NumPy and SciPy numerical and scientific libraries.

2.2.4 Word Embbedings techniques

Word embedding is a learned representation for text in which words which have the same meaning have a similar representation. Word embeddings are, in fact, a class of techniques where individual words are represented as real-valued vectors in a pre-defined vector space. Every word is mapped onto a vector and the vector values are learned in a way reminiscent of a neural network. The key to the approach consists of using a dense distributed representation for each word. This distributed representation is learned from the use of words. This allows words that are used in a similar way to have similar representations, by naturally capturing their meaning. In contrast, it can be compared to the crisp but fragile representation of a bag-of-words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

In this project,in order to be able to develop a stress text classifier, we will use different word embedding techniques trying to find one that provides the best response according to the available datasets. The techniques are:

- **Tf-idf**[16]: is a statistical measure that evaluates the relevance of a word to a document in a collection of documents. It is done by multiplying two metrics: number of times a word appears in a document and the inverse frequency of the word in a set of documents.

- **Word2Vec**: is a NLP technique that uses a neural network model to learn word associations from a large corpus of text. Once trained, the model is capable of detecting synonymous words or suggesting additional words for a partial sentence. This model represents each word with a carefully chosen vector in a way that a simple mathematical function would show the level of semantic similarity between the words represented by those vectors.

- **CountVectorizer**: is a tool provided by the Scikit-Learn library in Python. It is used to transform a given text into a vector based on the frequency of each word appearing in the entire text. It resembles to Tf-Idf but its behavior is more simple since it does not take into account the inverse frequency of each word in a set of documents.

- **Keras Embedding Layer**: Keras provides an embedding layer that can be used for neural networks on text data requiring the input data to be encoded with integers so that each word must be represented by a single integer number. This can be used to learn a single word embedding that can be saved and used in another model later on.

- **Pre-trained Word Embeddings**: are the embeddings learned in one task that are used to solve another similar task. These embeddings are trained on large data sets, stored, and subsequently used to solve other tasks. Thus, pre-trained word embeddings are a form of transfer learning. There are different types of pre-trained word embeddings: FastText, Word2Vec or GloVe.

  In this project, GloVe is going to be used. It is an unsupervised learning algorithm to obtain vector representations of words. It is trained on global co-occurrence statistics of words from a corpus, and the resulting representations show interesting linear substructures of the vector space of words.

2.2.5 Flutter

Flutter is an open source SDK for mobile application development made by Google. It is often used to develop user interfaces for Android, iOS and Web apps as well as a primary method for creating apps for Google Fuchsia.

Flutter is supported by four main pillars, its four main features, which are:

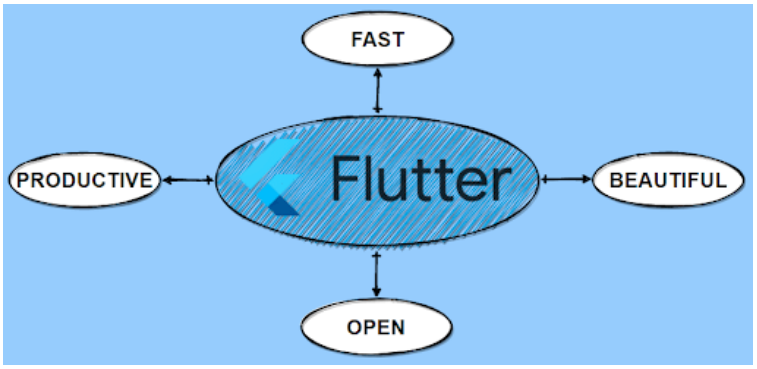

Figure 2.3: Flutter main pillars

- **Fast**

  Flutter allows us to develop applications faster by providing us with a set of tools in order to speed up both the development and the performance of the application.

- **Open**

  Both Flutter and Dart are OpenSource. But also, at the same time, we can say that the Flutter community is very open, which means that a large amount of documentation is constantly being generated thanks to Google's contributions combined with those of the rest of the developer community. All this work by the community, causes that the popularity of this SDK keeps increasing.

- **Beautiful**

  With Flutter we can design highly customized applications with interfaces from very simple to very colorful and expressive.

- **Productive**

  Flutter will allow us to generate multiplatform applications (for mobile, desktop and web) as we have said before.

While some skeptics did not believe that Flutter would be much different from other cross-platform development tools such as React Native or Xamarin, nowadays, we can see multimillion corporations were adopting this technology.

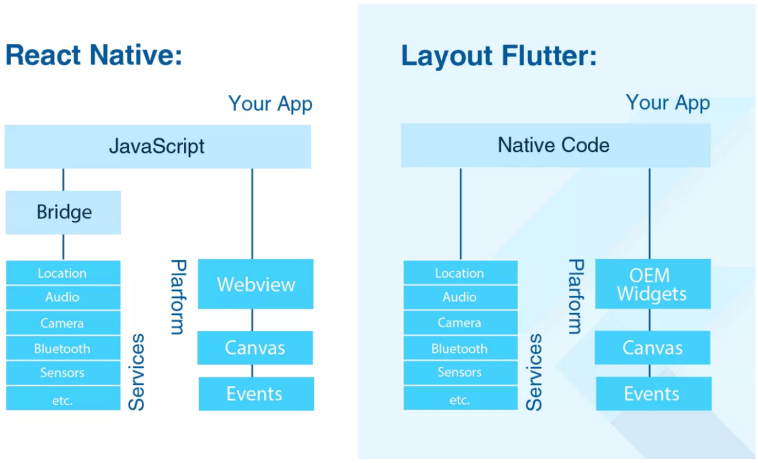

Figure 2.4: Comparison between Flutter and React

Unlike its cross-platform counterparts, Flutter is a mobile app SDK that seamlessly integrates with Android and iOS platforms to create appealing and customizable apps. Furthermore, Flutter does not require a bridge into the native platform layer to render UI with platform-specific UI SDKs.

CHAPTER 3

Stress-text detection

In this chapter, we will address the project's initial objective: the design of a machine learning model able to detect stress from text. The evaluation of the proposed model will also be discussed along with the analysis of the obtained results.

3.1 Datasets

The datasets that will be used to train the different Machine Learning models has been provided by Dreaddit [9] work and Genta [10] studies. As they are labeled datasets, we will train supervised learning models since it is a technique to deduce a function given a trained data.

3.1.1 Preliminary Analysis

A dataset analysis is then performed in order to identify the structure of the data and establish an approach to handle it. First of all, the labels of the datasets are checked to determine if there is any balance. We can observe in Fig. 3.1 a clear equilibrium in Dreaddit's labels very different from what we observe in Genta labels. However, according

to the paper [10] conclusions, we do not consider this as a significant factor to be taken into account.

As we are dealing with binary labeled datasets, the text with a label 1 will be considered as stressed text, the rest of the data will be labeled as 0.

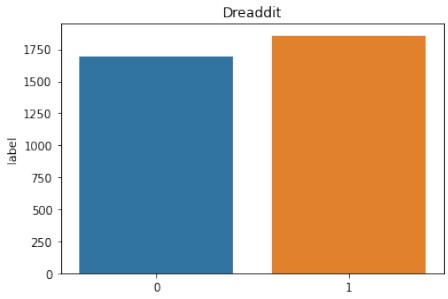

Figure 3.1: Labels for Dreaddit Dataset

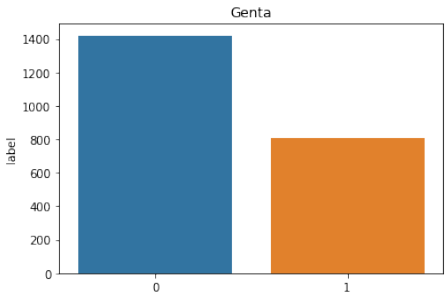

Figure 3.2: Labels for Genta dataset

Besides checking the equilibrium, we have also verified that there are no null values or values different from 1 or 0 on the labels. Afterwards, text composition has been analyzed in an attempt to visualize differences between stressed and unstressed text. Counts of words per post, characters and unique words (words that are not repeated within each post) have been made.

	Dreaddit		**Genta**	
	Stress labeled text	**Non stressed labeled text**	**Stress labeled text**	**Non stressed labeled text**
Words per	88.84	82.16	17.45	15.55
Characters per	459.69	435.10	86.85	77.14
Unique words per	66.30	63.07	15.22	13.46

Table 3.1: Differences between stressed and unstressed text

As it can be seen in Table. 3.1, texts with stress tend to be slightly longer and more complex than those without stress. In order not to limit ourselves only to these observations, we will employ more complex approaches such as word embeddings. As we presented before, word embedding is a machine learned representation for text where words that have the same meaning have a similar representation. This approach for words and documents representation may be considered one of the main advances of deep learning in NLP problems and we will use it attempting to enhance the performance of classifiers, in order to achieve better stress text recognition.

3.1.2 Text Preprocessing

Once the datasets have been analyzed, text is preprocessed in order to adapt it for later introduction as an input for a machine learning model. We define a set of functions that, through regular expressions, clean the text from special characters, stop words and english contractions. Afterwards, in order to improve the performance of the models, two different methods exist: Stem and Lemmatization. Stemming is the process of producing morphological variants of a root/base word. While lemmatization is the process of grouping the different inflected forms of a word so that they can be analyzed as a single element. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. This will reduce the size of the corpus and will help the classifiers on looking for related in it.

Word	**Stemming**	**Lemmatization**
information	inform	information
having	have	have
am	am	be
computers	comput	computer

Table 3.2: Lemmatization and Stemming Examples

We are going to use lemmatization as it is similar to stemming, but it gives context to the words. This way, words with similar meanings are merged into a single word facilitating the subsequent learning process. The resultant preprocessed text will be referred as "Clean Text" as we can see in Table 3.3. This is where we will apply the different word embedding techniques in order to be able to introduce them into the models with the aim of evaluating the presence of stress.

Text	**Clean text**
I can´t wait to go home and see him	cant wait go home see
I'm /@ cleaning 123 ¨ * this text for an example	clean text example

Table 3.3: Clean Text Example

3.2 Stress analysis models

We will now introduce the stress text analysis models proposed in our work. These models have been trained and validated in Genta and Dreaddit datasets. We describe the developed analyzer as well as the results obtained after using the different word embedding techniques.

3.2.1 Classifier

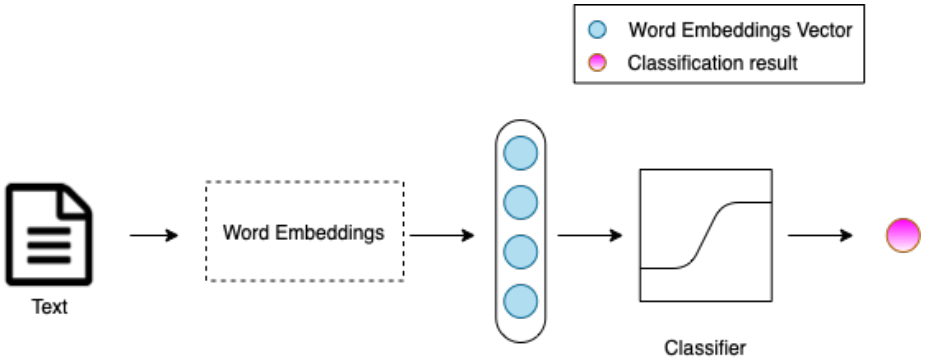

Figure 3.3: Stress Text Analysis model representation.

Once the text has been preprocessed, it will be embedded using different word embedding techniques in order to produce a vector. This vector will vary according to the technique used and will feed several classifiers. This project will use, in a first study, Tf-Idf, Word2Vec and Vectoricer as word embedding techniques.

Tf-idf measures how often a term or phrase appears within a given document, and compares it to the number of documents that mention that term within an entire collection of documents. Similarly to the way tf-idf works, Vectorizer is used to transform a given text into a vector based on the frequency of each word appearing in the entire text. But its behavior is more simple since it does not take into account the inverse frequency of each word in a set of documents. On the other hand, Word2Vec is not a singular algorithm, rather, it is a family of model architectures and optimizations that can be used to learn word embeddings from large datasets. Embeddings learned through Word2Vec have been shown to be successful in a variety of downstream natural language processing tasks.

These techniques will be introduced in a Logistic Regression classifier and in a Naive-Bayes Classifier. Subsequently, we will use keras embedding layer, pre-trained word embeddings such as GloVe and we will develop a convolutional neural network.

3.3 Experimental study

This section states the experiments conducted in order to to obtain a classification method capable of detecting stress as accurately as possible. Each experiment is made with two different datasets widely use for Sentiment Analysis projects. The metric used in this work is weight-averaged F-score. Accuracy, Precision and Recall are also calculated for all experiments.

At first, we entered the vectors generated by the Vectorizer, Tf-Idf and Wor2Vec techniques in a logistic regression classifier and in a Naive-Bayes classifier.

Logistic regression is a classification algorithm, used when the value of the target variable is categorical in nature. Logistic regression is most commonly used when the data in question has binary output. On the other hand Naive Bayes classifier is a probabilistic classifier based on Bayes' theorem and some additional simplifying assumptions. It is because of these simplifications, which are usually summarized in the hypothesis of independence between predicting variables.

After having trained the previously mentioned classifiers, we used the Word Embedding technique that provided us the best results, Tf-Idf as an input to a Keras designed neural network.

Keras supports two main types of models: The Sequential model API which is the one that we have used, and the functional API which can do everything of the Sequential model but it can be also used for advanced models with complex network architectures. The sequential model is a linear stack of layers, where the wide variety of layers available in Keras could be used. The most common layer is the Dense layer, which is the normal layer of a densely connected neural network with all its weights and biases. By this we aimed to achieve some improvement on previous models. But, before building the model, we needed to know the input dimension of the feature vectors. This occurs only in the first layer, as subsequent layers can do automatic shape inference.

We used binary cross entropy as loss function, Adam optimizer because it has a good performance in this type of problems, and a Sigmoid function as an activation function since it is a binary classification problem. After compiling it gave us a total of 88901 parameters for both layers for Dreaddit dataset, and 20321 parameters for Genta.

We specified a run of 100 epochs in order to see how the training loss and accuracy are changing after each epoch and the batch size. Batch size is responsible for how many samples we want to use in an epoch, which means how many samples are used in a forward/backward

pass. Since we have a small training set, we kept this to a low batch size (in our case 16).

Then, we changed the focus of the analysis and stopped using the vectors generated by Tf-idf. The reason is that we wanted to find a word embedding technique that provides better results than those previously seen. For this we started using Tokenizer, which can vectorize a corpus of text into a list of integers. Each integer corresponds to a value in a dictionary that encodes the entire corpus, with the dictionary keys being the vocabulary terms themselves. The indexing is ordered after the most common words in the text.

At this point, our data was still hardcoded. We did not allowed Keras to learn a new embedding space through successive tasks. So we used the Embedding Layer of Keras which takes the previously calculated integers and maps them to a dense vector of the embedding. With the Embedding layer we had, then, a couple of options. One way would be to take the output of the embedding layer and plug it into a Dense layer. In order to do this a Flatten layer was added in between so it prepared the sequential input for the Dense layer. By performing this, we increased the size of our training parameters. These embedding layer weights were initialized with random weights and then adjusted by back-propagation during training. This model takes the words as they came in sentence order as input vectors.

Although the accuracy obtained was better than the previous model, this was an unreliable way of working with sequential data. When working with sequential data, it is wise to focus on methods that look at local and sequential information rather than absolute positional information. This is why another approach to the problem was made by using a MaxPooling1D/AveragePooling1D layer or a GlobalMaxPooling1D/GlobalAveragePooling1D layer after embedding in order to reduce the size of the incoming feature vectors.

Alternatively to the previously discussed analyses, a precomputed embedding space using a much larger corpus was to be used. It is possible to precompute word embeddings by simply training them on a large corpus of text. For this purpose, GloVe from the Stanford NLP Group were be used, since its size was more manageable than the Word2Vec word embeddings provided by Google. After a quick check, we observed that, for Dreaddit, 94.9% of the vocabulary was covered by the pre-trained model, which was a good coverage of our vocabulary. However, the same cannot be said for Genta, as only 22% of the vocabulary was covered. But we had not trained our word embeddings, so we expected the performance to be lower. This changed by allowing the embeddings to be trained but did not enhace the model as much as we wanted. This method is useful for large training sets since it can push the training process to be much faster than without it. Here it seems to help, but not much. However, this did not have to be due to the pre-trained word embeddings.

Therefore we will conclude the study by focusing on a more advanced neural network

approach: CNN also known as Convolutional Neural Networks. With this we wanted to see if it was possible to enhance the model and give it the edge over previous techniques.

CNN could be understood as a specialized neural network capable of detecting specific patterns. It has hidden layers called convolutional layers which consist of multiple filters that slide through the data and that are able to detect specific features. That is the mathematical process of convolution. Each convolutional layer makes the network capable of detecting more complex patterns. For this analysis, since we are going to analyze sequential data, the CNN will be unidimensional.

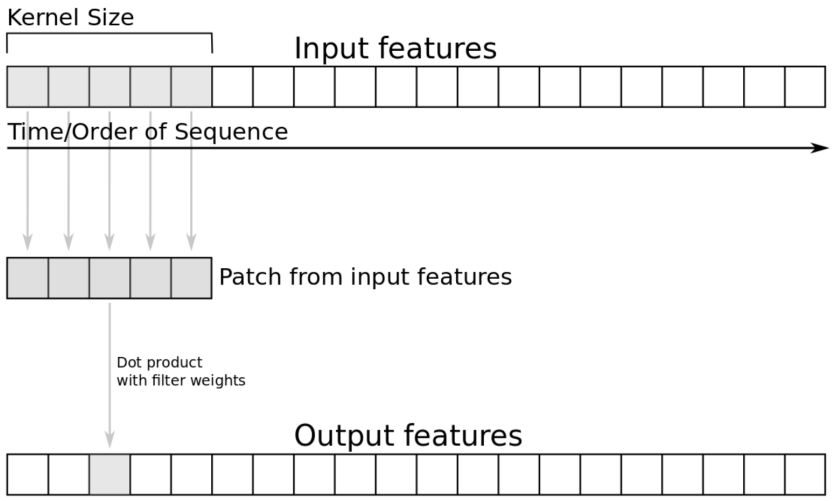

Figure 3.4: Uni-dimensional convolution

Fig 3.4 tries to represent how a convolution of this type works. It starts by taking a patch of input features with the size of filter kernel. Using this patch, the dot product of the multiplied filter weights is taken. One-dimensional CNN is invariant to translations, which means that certain sequences can be recognized at a different position. This can be useful for certain patterns in text.

Keras offers several convolutional layers that can be used for this task. In this case, the layer that was used, was the Conv1D layer. This layer again has several parameters to choose from but we only used the number of filters, kernel size and activation function. Also this layer was placed between the Embedding layer and the GlobalMaxPool1D layer.

3.4 Results

In this section, experimental results are shown and discussed. The experimental results of the proposed models are collected in Table 3.5. In order to understand the results obtained, some concepts must be previously defined.

- **Confusion Matrix**: is a specific table layout that allows visualization of the performance of an algorithm. It is made up of 4 different combinations of predicted and actual values.

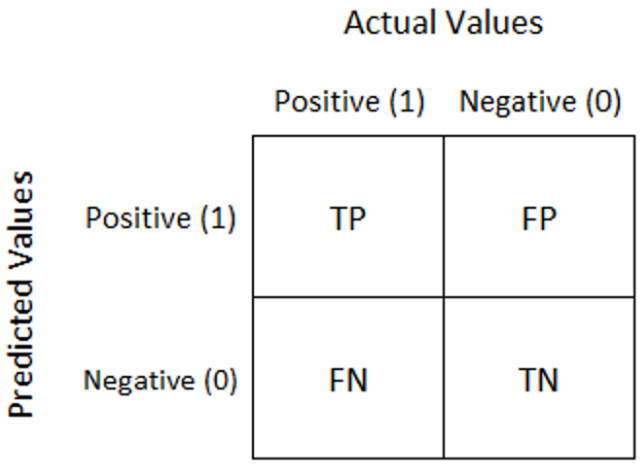

Figure 3.5: Confussion Matrix

We will now see what does each combination of values means:

- **True Positive**: is the case where the model predicts correctly a positive label. In our case, the model will predict as stressed, the text labeled with a 1.
- **True Negative**: happens when a negative label is predicted correctly.
- **False Positive**: referring to our model, is when it predicts stress when the data has no signs of it. This is also known as type 1 Error.
- **False Negative**: occurs when the model could not find stress within a text marked as stressed. This is consider as a Type 2 Error.

We can combine the above concepts in order to produce metrics that will help us to analyze and evaluate the model. These metrics are:

- **Precision**: measures how many of our predictions, are actually true positives.

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

- **Recall**: analyzes how good is the performance of the model identifying true positives.

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

- **Accuracy**: measures how many stress detections have been correctly performed among all samples.

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

- **F1-score**: helps to measure recall and precision simultaneously using the harmonic mean instead of the arithmetic mean.

$$F1 = \frac{2 * Recall * Precision}{recall + Precision}$$

After performing a logistic regression on the vectors resulting from the Vectorizer and Tf-Idf techniques we were able to see very similar results for both techniques. In contrast, the accuracy obtained with Word2Vec was far lower (for both datasets), for this reason it was discarded and was not be applied to the rest of the models. This is because Wor2Vec has a superior performance, if trained with much larger datasets. Next, in order to achieve better accuracy, the two remaining techniques were introduced in a Naive-Bayes classifier.

With Naive-Bayes classifier, we improved the accuracy obtained by the logistic regression models and achieving the highest f-score of the entire study: 76%. This value is slightly higher than the one obtained by the same method with the Dreaddit dataset, which indicates the great potential of this model.

	Actual stress	
Predicted	252	27
stress	101	67

Table 3.4: Confussion matrix for Naive-Bayes classifier using Tf-Idf word embedder.

Nevertheless, a more exhaustive analysis had been carried out using Tensorflow, the embedding layer of keras, GloVe and Convolutional Neural Networks (CNN) on both datasets.

We can verify, for Neural Network with Tf-Idf vector as input, that in this case, the test and validation sets are both too small. Consequently, the performance of the neural network

was not as expected and overfitted the training model. This happens because (deep) neural networks perform better when we have a large number of samples.

After using GlobalMaxPooling1D layer, a significant improvement in the performance of the Keras model with a dense layer, was observed reaching an accuracy of 71% (by using Dreaddit). Nevertheless, the Naive-Bayes model for Tf-Idf keeps providing the best performance.

		Dreaddit			**Genta**		
Method	**Word embbeding**	**P**	**F**	**R**	**P**	**F**	**R**
Log. Reg	Vectorizer	0.72	0.72	0.72	0.72	0.71	0.72
NB		0.74	0.72	0.72	**0.76**	**0.76**	**0.76**
Log. Reg	Tf-idf	**0.74**	**0.74**	**0.74**	0.72	0.70	0.72
NB		0.72	0.63	0.66	0.71	0.69	0.71
Log. Reg	Word 2 Vec	0.57	0.55	0.57	0.62	0.58	0.57
Keras NN	Tf-idf	0.67	0.67	0.67	0.70	0.70	0.70
Dense Layer	Vectorizer	0.71	0.71	0.71	0.69	0.69	0.70
MaxPool1D		0.71	0.70	0.70	0.75	0.73	0.74
Glove not trained	Glove	0.62	0.61	0.62	0.70	0.69	0.70
Glove trained		0.67	0.67	0.67	0.74	0.73	0.74
CNN	Vectorizer	0.71	0.71	0.71	0.71	0.71	0.71

Table 3.5: Precision (P), recall (R) and F-score (F) for stress text analysis classification models.

Finally, after analyzing the results, we saw that 71% accuracy seems to be a difficult hurdle to overcome with these datasets and a CNN may not be well equipped. This could be because we do not have enough training samples, we had data that did not generalize well or lack focus in hyperparameter fitting. CNNs work best with large training sets where they are able to find generalizations where a simple model such as logistic regression will not be able to.Despite this, an F-score of 76% was achieved for the Naive-Bayes model trained

with the Tf-Idf word embbedding. This is a great result since our classifier will be able to detect stress in more than three out of four texts it analyzes.

CHAPTER 4

Keystroke Dynamics Stress Detection

This chapter covers a new stress detection technique that aims to identify stress through the keystroke dynamics of a subject. It is intended to complement the technique seen in the previous chapter by providing higher levels of confidence when recognizing the presence of stress in an individual.

The evaluation of the proposed model along with the analysis of the obtained results will be discussed.

4.1 Dataset

Different Machine learning models will be trained using the dataset provided by the SWELL project [11]. SWELL is a labeled dataset resulting from the stress analysis of individuals using non-invasive techniques inside work environments. Moreover, like Dreaddit, we will train supervised learning models due to the fact that we will be dealing with a labeled dataset.

4.1.1 Cleaning Process

SWELL project not only uses keystroke dynamics as stress identifiers, but also addresses non-invasive techniques such as posture recognition, facial expression detection, heart rate analysis, etc. For our analysis will proceed to clean the dataset of all those features irrelevant to our study.

We will clean the dataset in a similar way as we did for text analysis. Firstly we will discard those features that are not relevant for our study and we will look for Nan or null values in the desired features for their subsequent elimination. Once we have the desired dataset, we observe that labeling is not binary. Therefore, we will transform it using the median as threshold. This method guarantees, as can be seen in Fig 4.1, a balanced dataset.

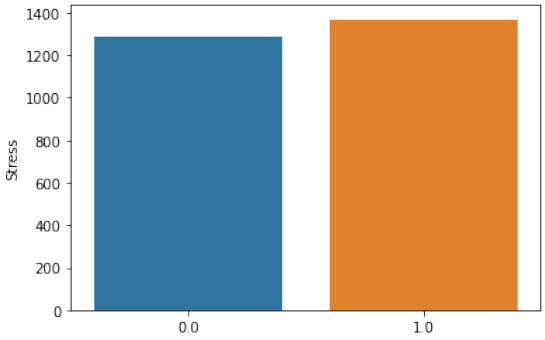

Figure 4.1: Labels for SWELL Dataset

4.1.2 First Analysis

During this cleanup we have found certain features related to mouse usage that might provide significant information in web environments. Subsequently, and similarly to what was done for text analysis, each feature will be analyzed in an attempt to observe any variation in presence of stress. Data obtained is presented in Tab 4.1 below.

Notice how the data presented above show a slight increase when the individual is not under stress for nearly all of the features. Even though we can extract some conclusions at a glance from the data obtained, it would not be enough to establish whether the individual is under stress or not. As a consequence, these features will be introduced in different Learning Models that will be used to define whether a person is stressed or not more accurately.

Features (per minute)	Stress	Non-stress
Mouse events	15.47	16.07
Nº of left clicks	7.12	6.99
Keystrokes	71.54	91.31
Characters	49.67	57.66
Error Keys	8.32	10.30
Special Keys	29.87	36.65
Shortcuts	0.68	0.87
Spaces	10.35	12.02
Character Ratio	0.49	0.52
ErrorKey Ratio	0.18	0.25

Table 4.1: Keystroke dynamics stress behaviour

4.1.3 Classification models

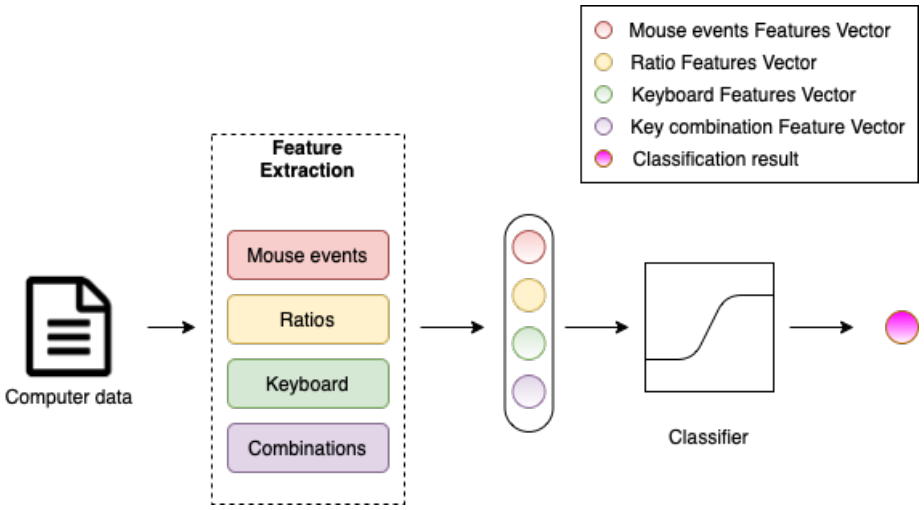

Figure 4.2: Keystroke Dynamics model representation

Once the preliminary study has been carried out, the dataset will be divided into a training dataset and a test dataset in order to train the classifiers; with the training dataset consisting

of the 80% of the original data (2125 samples) and the test dataset being the remaining 20% (532 samples). All features related to the use of the computer have been grouped according to the events they represent: mouse use, error key ratio or character ratio, key combinations and keyboard elements. Due to the high number of features available, we will use RFECV tool provided by Sklearn. This tool removes features recursively in order to use an optimal number that ensures the most accurate results after training the model. This is based on validation scores and the results are collected in Table 4.2.

These features will be used to train a Logistic Regression model and the Naive-Bayes model which were previously used for the text analysis. In addition, a Support Vector Classification (SVC) model will be trained. We have choosen SVC due to its great performance in other projects like [16] and [11].

Optimal Nº of Features for each model		
Model	Number of features	Features
LR	4	Error key ratio Mouse activation Shortcuts Character ratio
SVC	4	Error key ratio Mouse activation Shortcuts Left click
LR	8	Error key ratio Mouse activation Left click Error keys Special keys Shortcuts Spaces Character ratio

Table 4.2: Optimal features for each model

4.2 Results

Once the optimal number of features for each model has been selected, we proceed to train them resulting in the following table. The Support Vector Classification (SVC) technique is the one that has provided us with the highest accuracy, with an accuracy of 0.59 and an F1 score of 0.62, so it will be the one we will use in our application. It is important to highlight the improvement provided by the feature selector to the model in terms of performance. It increases the f-score of all models by almost 10%.

	Actual stress	
Predicted stress	78	41
	179	234

Table 4.3: Confussion matrix for SVC using feature selection.

Due to the low accuracy of the model, it will only be used to support the stress classifier in the text. Considering the small size of the data set and the further studies mentioned in the previous chapter, it is not worthwhile to introduce these data into a neural network or to use deep learning techniques, so no further study will be carried out.

	With feature selector			**Without feature selector**		
Method	**P**	**F**	**R**	**P**	**F**	**R**
SVC	**0.73**	**0.62**	**0.59**	0.64	0.55	0.52
NB	0.56	0.56	0.56	0.46	0.46	0.46
LR	0.62	0.58	0.57	0.54	0.52	0.51

Table 4.4: Precision (P), recall (R) and F-score (F) for the proposed supervised models.

CHAPTER 5

Stress-detection App

In this chapter, we want to develop an application that merges the two models that provided the best results in previous chapters into an application. The idea is to design a simple and user-friendly application that could be deployed in work or academic environments. Besides being deployed in academic or work environments, it could be integrated into existing communication platforms in those environments. Thus, Flutter SDK has been chosen as it allows to develop UIs for Android, iOS and Web applications.

5.1 Architecture

As you can see in Fig 5.1, the general architecture of the system is made up of 2 blocks: the application and the stress detector system (SDS) that includes the Stress Detector.

The main objective of the application is to detect stress in a user through his typing patterns and the typed text. To do this, a listener will record this data and send it to the SDS. The SDS that receives this data is composed of the two machine learning models developed for this project and a detector. The detector will determine which machine learning model should act at any given moment to return the best possible prediction to the application.

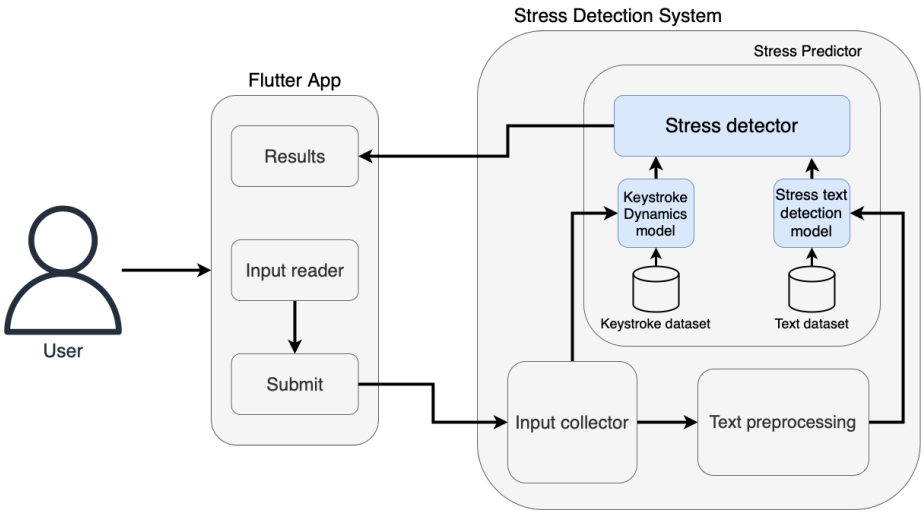

Figure 5.1: System architecture

5.2 Stress Detection System (SDS)

The stress detection system or SDS is in charge of processing and returning a response based on the information provided by the listener.

As we have seen previously, Machine Learning models do not recognize text by itself as an input variable. For this reason this text will be vectorized allowing the information to be sent to the stress text detection model. Once an answer has been obtained, it will be returned to the application in the form of an image with the prediction written below.

The **input collector** gathers all the data provided by the listener and stores them in two variables, one containing the text that is going to be vectorized and the other containing the keyboard patterns that is inserted directly into the predictor.

The **text preprocessing** block, collects the variable and process it using the Tf-Idf word embedding technique which, as seen in previous chapters, is the one that has given us the best results.

The collected data will be inserted into the **Stress Detector** in order to obtain a result after analysis. This is the main module of the system since it is in charge of stress detection. For this purpose, different models have been trained using specific datasets according to each detection technique. Once we have obtained satisfactory training results, the detector will be ready to send the results to the application.

Finally, once the prediction is obtained, the Stress Detector will be in charge of returning a response to the application. The Stress detector will mark as **0 or no stress** in case it

does not detect stress, otherwise it will mark the data with **1 or stressed**. In both cases the application will receive the data and return an image and a text with the result of the prediction.

5.3 Application

The Application is a key block in the project as it allows the communication between the user and the SDS. The main feature is a Listener that collects user's typed text and records his/her keystroke patterns.

Once the user wants to analyze his stress level, he will press the submit button present in the application. This button will send all the collected information to the stress predictor, where a response will be elaborated according to the input. The application is mainly composed of a Listener and a submit button.

As previously mentioned, the **Listener** will be responsible for recording everything that the user writes and the way it is written. To develop it, we will use the RawKeyboardListener widget provided by flutter.

The **Submit button** is in charge of sending the information collected by the listener to the stress predictor, so it could later provide the results of the analysis.

5.3.1 Application Design

The main idea is to create an application with an UI very similar to Google translator but, when clicking on the compile button, rather than translating text, it analyzes the stress present in the text and registers the typing patterns while it is being typed. In order to do this, it is necessary to understand how Flutter works. Flutter groups everything into sub-Widgets, for example, a column widget for stacking elements upright, padding widgets for adding extra space, text widget for labeling, and so on. The only thing required is a design that allows the widgets to stack neatly.

Our design will consist mainly of a body and a floating button. In the body we will house the 3 most important elements of the design: an input text box, the submit button and a wrapper that will show the results from the prediction stacked one on top of the other. The floating button will be used as a button that will provide help to the user.

For the first version of the application, we decided to go for a simple design that reflects the main purpose of the application which, after all, is what gives it added value. It will

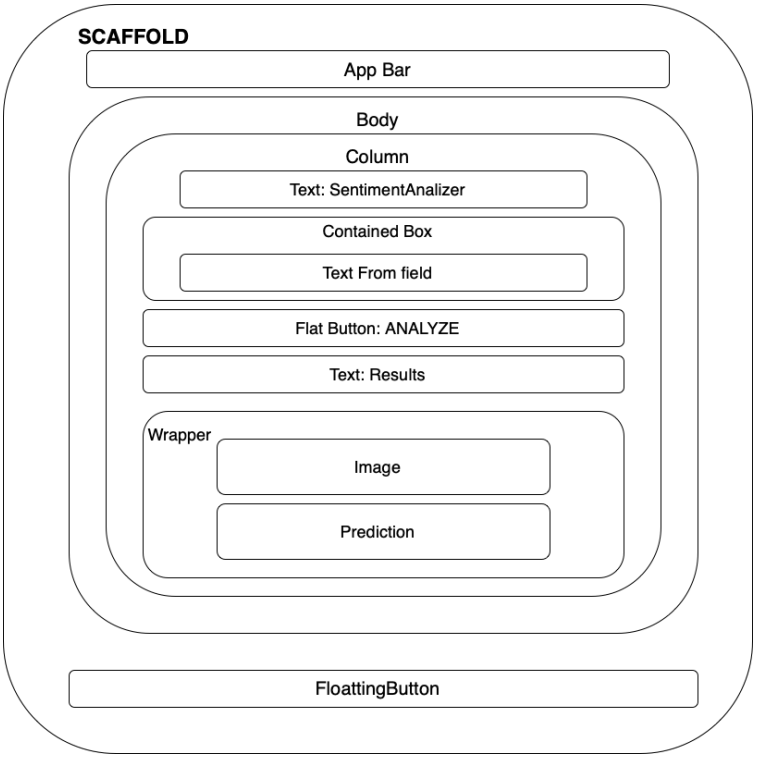

Figure 5.2: Visual Representation of Flutter code/layout

mainly consist of 2 parts: a Text Box where we will place our listener which will transfer the data to the Machine Learning models in order to record users' typing patterns and analyze the stress found in the text, and a table where we will show the prediction followed by an image. It is important to mention that the Keystroke Dynamics model will receive a continuous supply of data provided by the listener that will record any activity related to the keyboard and the mouse. Moreover, the Text Analysis model will only be activated when the 'Analyze' button is pressed as shown in Fig 5.3 and Fig 5.4.

As previously mentioned, one of the great advantages of Flutter is the ease of creating a UI which could be deployed not only in web environments but also in iOS and Android environments. Fig 5.6 shows an example of how the application would look deployed in an iOS environment. It should be noted that the operation is the same as in a web environment because the listener will not only record keystrokes as it would do for a computer keyboard, but it will also record the taps on the screen and the sliding on the same as if it were a mouse. Consequently, Keystroke Dynamics' model will not lose accuracy in mobile environments.

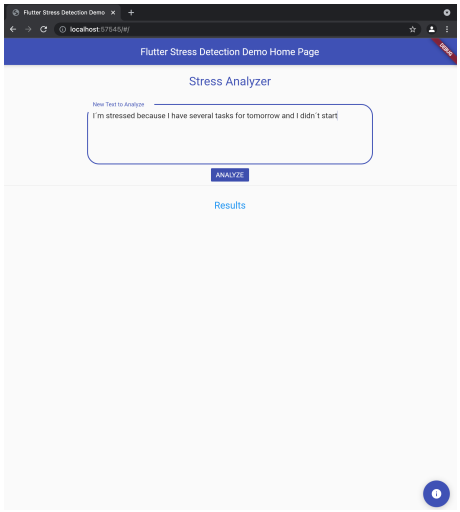

Figure 5.3: App before pressing 'ANALYZE'

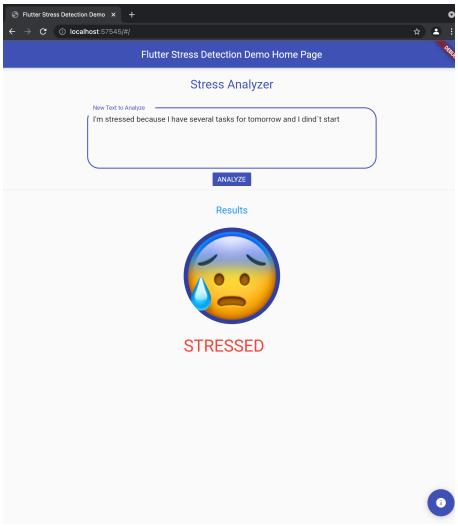

Figure 5.4: App after pressing 'ANALYZE'

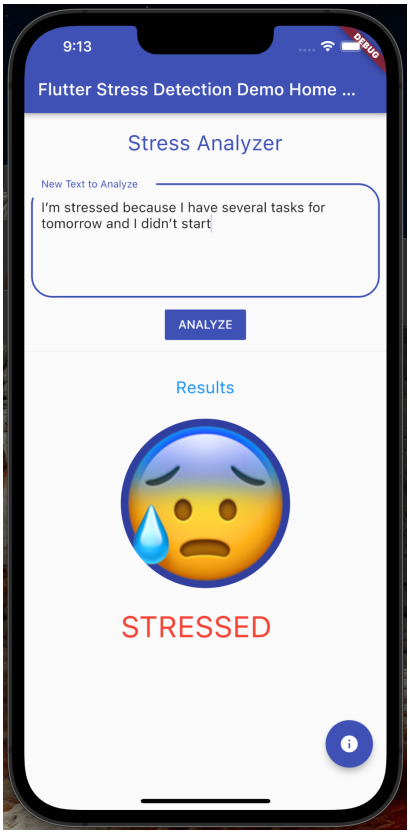

Figure 5.5: iOS App deployment

5.4 Execution Process

This section shows an example of how the system works based on the defined architecture.

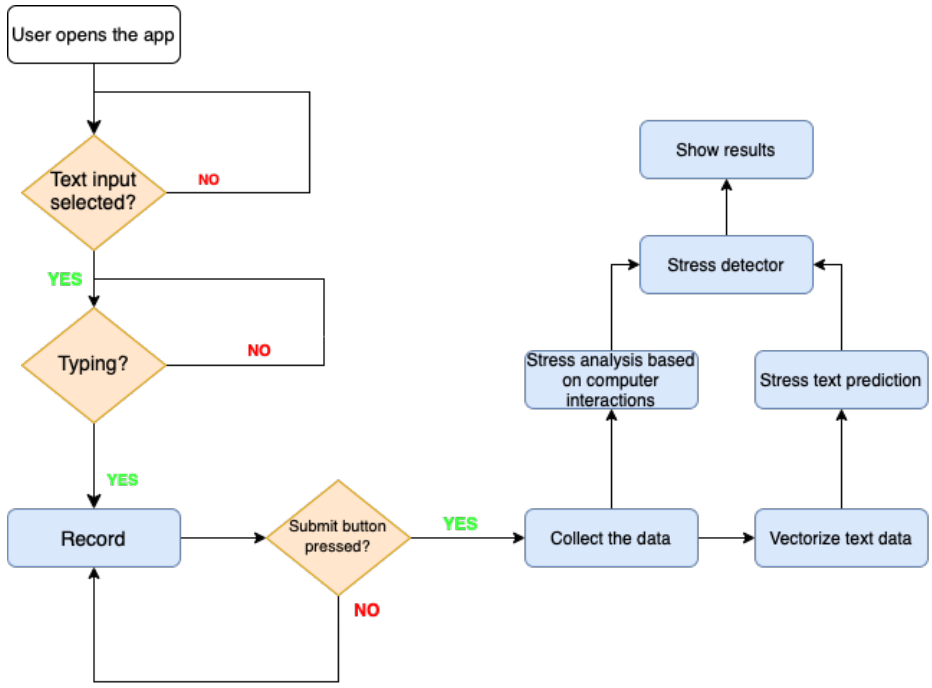

Figure 5.6: Decision diagram of the execution cycle

The user will open the application from is browser or even from any Android or iOS device. Here, the listener will be located in the text entry. The listener will be activated when the user accesses the text box, recording all the actions performed and collecting the text typed in. This data collection process will not stop until the user clicks on the submit button marked as 'ANALYZE', which will cause the data to be sent to the SDS. The data will be processed and a result of the analysis defined by the decisor will be returned to the application.

CHAPTER 6

Conclusions and future work

This chapter will describe the conclusions drawn from the project, as well as the results obtained and some of the problems encountered with their respective solutions. We will conclude with a few ideas for future work.

6.1 Conclusions

In this section, all conclusions drawn during each stage of the project are presented.

Following a study of the state of the art in stress detection, two non-intrusive models have been defined. One model will try to detect stress from text and the other from pulse dynamics. Three datasets have also been obtained which, after preprocessing, have been introduced in the classifiers. For this purpose, an experimental study has been designed where two models have been implemented using different techniques in order to evaluate their performance on the chosen datasets.

The first conclusion we draw from the analysis arises from the text prediction models. After training them with different datasets, we have obtained a performance higher than 76%. This shows that it is possible to detect stress text in a pretty confident way providing

significant value to the application. Then, we trained the Keystroke Dynamics classifiers obtaining a performance of 62%. This is a really good performance since we have a small and unusual keystroke dynamics dataset. Since the accuracy of this model is lower than the previous one, it will be used as a decision maker in case of lack of accuracy in the previous model. Finally, we have managed to develop a fully functional application capable of detecting a user's stress accurately by merging these two different techniques.

6.2 Achieved goals

This section details the objectives that have been achieved throughout the project.

Collection and preprocess of datasets so they can be used to train the desired models.

The first objective was to find a datasets capable of detecting stress in a subject by using non-invasive techniques such as text stress analysis, or by analyzing the computer interactions. For this purpose SWELL, Dreaddit and Genta datasets were acquired from a previous research, and will be used according to the technique that will be developed. Before training the models, we had to adapt the content of the datasets. In case of Genta and Dreaddit datasets, data was preprocessed in order to clean the text of unwanted characters for later tokenization. On the other hand, SWELL dataset had to be balanced and the 'Nan' characters were removed from it to facilitate the future analysis.

Definition of a Machine Learning model capable of identifying stress in text.

Once Dreaddit data was preprocessed, different Word Embedding techniques were studied and a variety of Machine Learning algorithms were applied in order to acquire a model capable of accurately predicting the presence of stress in text. Finally, Naive-Bayes Classifier with Word Embedding Tf-Idf was the best performing model, something expected since these two techniques perform very well with small datasets and binary classification problems.

Definition of a Machine Learning model capable of identify stress based on typing patterns.

After cleaning and balancing SWELL dataset, different classification models were studied, not without previously using a technique that enabled us to obtain the number of features that would optimize the performance of these classifiers. SVC achieved the best results with a 60% accuracy rate using only 4 features.

Development of a working tool that integrates the previous models.

We finally developed an application with the two previous models included in it. This application will not only be deployed on web environments but will also be responsive on iOS and Android environments; reaching a larger number of devices.

6.3 Problems Faced

This section details the problems that have been occurring during the course of the project.

- **Lack of a unique dataset**

  The main problem we have encountered when developing this project has been the lack of a single dataset that combines Keystroke Dynamics and Text data. In addition, despite the fact that sentiment detection techniques in text are highly developed, there are not many tagged datasets used to identify emotions based on keystroke patterns.

- **SWELL dataset**

  Another major problem we have faced has been the data present in the SWELL dataset, since they are not the ones that would normally be used for a Keystroke Dynamics analysis. For this reason, the recursive feature elimination technique was applied in order to obtain the features that provided more information.

- **Learning Flutter**

  Finally, although its programming language (dart) is very similar to javascript, Flutter does not work in the same way. Therefore, we have invested some time in learning how to use this tool in order to be able to design the App correctly.

6.4 Future work

- **Creation of a new dataset**

  As previously mentioned, the creation of a data set that combines keystroke dynamics and stress text data would increase the performance of the application and would facilitate obtaining a model capable of predicting stress in a much more accurate way. However, if it is not possible to create such dataset, we have found that by increasing the size of the datasets, we can apply techniques that significantly increase the accuracy of the prediction.

- **Improving the application**

  We want to improve the application by introducing a login screen for user identification, as well as a persistence layer that allows the user to have a history of the texts and predictions that have been made over time. It is also intended to improve the design to give a more attractive image.

- **Allow user validation**

  In order to improve the performance of the classifiers, we intend to add a validation function whereby the user will be able to indicate whether the prediction is correct or not. This will allow the creation of a new labeled dataset.

- **Use other unobtrusive techniques**

  Several studies have shown that certain techniques provide very good results when it comes to detecting stress, for example, body posture recognition and facial expression analysis. These techniques could be carried out using a depth-sensing camera such as Kinect.

APPENDIX A

Impact of this project

This appendix reflects, quantitatively or qualitatively, on the possible social and economic impact as well as ethical implications.

A.1 Social impact

This section will discuss the social impact that this project could have.

This project focuses on one of the main causes of the majority of sick leaves and the main factor behind the decrease in worker performance, stress. This project aims to improve the well-being of users in work and academic environments by providing one that favors optimal results. In working environments, an employee who feels comfortable in his workplace, is more likely to stay in the company and not move to a competitor. Similarly, in academic environments, students will show greater interest in the subject in a de-stressed environment.

A.2 Economic impact

This section covers the potential economic impact that can be deduced after the realization of the project.

As mentioned above, a de-stressed work environment significantly improves the performance of the individual, which roughly translates as an increase in company profits or, if we are talking about academic environments, an improvement in academic performance. In addition, this stress detection technique uses exclusively a mobile device or a computer. These elements are available in work and academic environments, making its deployment much easier since it is not necessary to make an economic outlay to acquire sensors that allow such analysis.

A.3 Ethical implications

In this section we will discuss the ethical issues that may arise from our project.

The main ethical problem that the project presents would be the violation of privacy that the deployment of this tool would entail since the tool would monitor in real time everything that the user writes on his device. However, users would be notified about the benefits of the application and their consent would be required prior to implementation.

APPENDIX B

Economic budget

This appendix details the financial budget required to carry out this project. Physical resources and human resources are presented.

B.1 Physical resources

This section is intended to detail the estimated budget required relative to the hardware.

The entire budget included in this section is related to the personal computer used to carry out the project. Although it would have been enough to use any other equipment with lower processing capacity, we will now detail the characteristics of the one used for this project:

- **CPU:** Intel Core i7 2.70GHz x4
- **RAM:** 16GB
- **Disc** 250GB SSD

This laptop costs arround 900€.

B.2 Human resources

This section, in a very similar way to the previous one, will attempt to estimate the human budget.

For this purpose, it will be assumed that the entire project has been carried out by one person in a time equivalent to 12 ECTs; which gives a total time of 324 hours. Assuming an average of 6 hours of work per day, that each month has approximate 21 working days and considering that the average salary of a GSI scholarship is 500 €/month, we estimate a total salary of around 1300 €.

B.3 Conclusion

Therefore, the economic costs associated with the creation of this project amount to €2200.

Bibliography

[1] Schneiderman N, Ironson G, and Siegel SD. Stress and health: psychological, behavioral, and biological determinants. *Annu Rev Clin Psychol. 2005;1:607-28. doi:*, 10., 2005.

[2] Juliet Hassard, Kevin Teoh, Gintare Visockaite, Philip Dewe, and Thomas Cox. The cost of work-related stress: a systematic review. *Journal of Occupational Health Psychology*, 23, 03 2017.

[3] TY JOUR AU Frantz, Anna AU Holmgren, and Kristina PY. Da - 2019/11/27 TI - The Work Stress Questionnaire (WSQ) – reliability and face validity among male workers JO - BMC Public Health SP - 1580 VL - 19 IS - 1 AB - The Work Stress Questionnaire (WSQ) was developed as a self-administered questionnaire with the purpose of early identification of individuals at risk of being sick-listed due to work-related stress. *It has previously been tested for reliability and face validity among women with satisfying results. The aim of the study was to test reliability and face validity of the Work Stress Questionnaire (WSQ) among male workers. SN* -, 1471-2458, 2019.

[4] Shalom Greene, Himanshu Thapliyal, and Allison Caban-Holt. A survey of affective computing for stress detection: Evaluating technologies in stress detection for better health. *IEEE Consumer Electronics Magazine*, 5(4):44–56, 2016.

[5] Md Fahim Rizwan, Rayed Farhad, Farhan Mashuk, Fakhrul Islam, and Mohammad Hasan Imam. Design of a biosignal based stress detection system using machine learning techniques. In *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, pages 364–368, 2019.

[6] G. Shanmugasundaram, S. Yazhini, E. Hemapratha, and S. Nithya. A comprehensive review on stress detection techniques. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, pages 1–6, 2019.

[7] Zhai J and Barreto A. Stress detection in computer users based on digital signal processing of noninvasive physiological variables. *Conf Proc IEEE Eng Med Biol Soc. 2006;2006:1355-8. doi:*, 10., 2006.

[8] Suranga D. W. Gunawardhane, Pasan M. De Silva, Dayan S. B. Kulathunga, and Shiromi M. K. D. Arunatileka. Non invasive human stress detection using key stroke dynamics and pattern variations. In *2013 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 240–247, 2013.

[9] Elsbeth Turcan and Kathy McKeown. Dreaddit: A Reddit dataset for stress analysis in social media. In *Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI 2019)*, pages 97–107, Hong Kong, November 2019. Association for Computational Linguistics.

[10] Genta Indra Winata, Onno Pepijn Kampman, and Pascale Fung. Attention-based lstm for psychological stress detection from spoken language using distant supervision. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr 2018.

[11] Saskia Koldijk, Mark A. Neerincx, and Wessel Kraaij. Detecting work stress in offices by combining unobtrusive sensors. *IEEE Transactions on Affective Computing*, 9(2):227–239, 2018.

[12] S. Rajarajeswari and Sowmya. C. *Drishya. K. Ramdas, Soumyashree Dhabade, Shirisha. M, Samyuktha H. R 2020. Cognıtıve Stress Detectıon Usıng Keystroke Dynamıcs And Pattern Varıatıons. International Journal of Advanced Science and Technology. 29*, 29(7):12036–12050, 2020.

[13] Guillem Aguado, Vicente Julián, Ana García-Fornes, and Agustín Espinosa. Using keystroke dynamics in a multi-agent system for user guiding in online social networks. *Applied Sciences*, 10(11), 2020.

[14] Lisa M. Vizer, Lina Zhou, and Andrew Sears. Automated stress detection using keystroke and linguistic features: An exploratory study. *International Journal of Human-Computer Studies*, 67(10):870–886, 2009.

[15] What is machine learning? a definition., May 2021.

[16] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing Management*, 39(1):45–65, 2003.

[17] Anthonette Cantara and Angie Ceniza. Stress sensor prototype: Determining the stress level in using a computer through validated self-made heart rate (hr) and galvanic skin response (gsr) sensors and fuzzy logic algorithm. *International Journal of Engineering Research Technology*, 5:28–37, 03 2016.