

# Classifying Disease-related Tweets in the Twitter Health Surveillance System

By

Cristian Camilo Garzón Alfonso

A thesis submitted in partial fulfillment of the requirements for the degree

of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

2018

Approved by:

---

Manuel Rodríguez Martínez, Ph.D.  
President, Graduate Committee

---

Date

---

Wilson Rivera Gallego, Ph.D.  
Member, Graduate Committee

---

Date

---

Pedro Rivera Vega, Ph.D.  
Member, Graduate Committee

---

Date

---

William Hernández  
Graduate School Representative

---

Date

---

José Colom Ustariz, Ph.D.  
Department Chairperson

---

Date

Abstract of Thesis Presented to the Graduate School  
of the University of Puerto Rico in Partial Fulfillment of the  
Requirements for the Degree of Master of Science in Computer Engineering

## **Classifying Disease-related Tweets in the Twitter Health Surveillance System**

Public health officials, hospital directors, and other professionals related with health disciplines have to track and report disease outbreaks that affect populations around the world. Often, the data comes in reports and Comma Separated Values (CSV) files from hospitals, and private doctor's offices. Typically, these reports are generated manually, increasing the risk of human error contained in transcript, analysis, charts, and different indicators that are used by professional organizations such as the United States (US) Center for Disease Control (CDC), World Health Organization (WHO) or US Health & Human Services (HHS). The processing and understanding of all these data might take weeks and the official warnings to a population could arrive too late. Poor and undeserved communities normally are highly affected since limited access to medical services often means that medical care attends the outbreaks when the major part of the community is already affected.

In this research we present the Twitter Health Surveillance (THS) application framework. THS is designed as an integrated platform to help health officials collect tweets, determine if they are related with a medical condition, extract metadata out of them, and create a big data warehouse that can be used to further analyze the data. THS is built atop open source tools and provides the following value added services: Data Acquisition, Tweet Classification, and Big Data Warehousing.

In order to validate THS, we have created a collection of roughly twelve thousands labelled tweets. These tweets contain one or more target medical terms, and the labels indicate if the tweet is related or not to a medical condition. We used this collection to test

various machine learning models based on Recurrent and Convolutional Neural Networks. Our experiments show that we can classify tweets with 96% precision, 91% recall, and 86% F1 score. These results compare favorably with recent research on this area, and show the promise of our THS system.

Resumen de tesis presentada a la Escuela Graduada  
de la Universidad de Puerto Rico como requisito parcial de los  
requerimientos para el grado de Maestría en Ciencia de Ingeniería de Computadoras

## **Clasificación de tweets relacionados con enfermedades en Twitter Health Surveillance**

Oficiales de salud pública, directores de hospitales, y otros profesionales relacionados con disciplinas del área de salud, tienen que proveer seguimiento y reportar brotes de enfermedades, que afectan a las poblaciones alrededor del mundo. Típicamente, estos reportes son generados manualmente, incrementando el riesgo del error humano en la transcripción, análisis, ilustración y diferentes indicadores que son usados por organizaciones profesionales como el Centro de Control de Enfermedades de los Estados Unidos de Norteamérica (CDC), la Organización Mundial de la Salud (WHO) o por el Departamento de Salud y Servicios Humanos de los Estados Unidos de Norteamérica (HHS). El procesamiento y entendimiento de toda esta data puede tardar unas semanas y las alertas de oficiales pueden llegar muy tarde a la población. Las comunidades más pobres y desamparadas normalmente están altamente afectadas debido a las limitaciones para acceder a los servicios médicos, y muchas veces esto significa que el personal médico atiende los brotes muy tarde, cuando la mayor parte de la comunidad ya está afectada.

En esta investigación presentamos Twitter Health Surveillance (THS) como una aplicación de referencia. THS está diseñada como una plataforma integrada para ayudar a los oficiales de salud en la recolección de tweets, determinando si estos están relacionados con una condición médica, extraer los metadatos y crear la bodega de grandes datos, que pueden ser usados para un futuro análisis de los mismos. THS está construido con herramientas de acceso libre y provee los siguientes servicios de valor agregado: adquisición de los datos, clasificación de los tweets y almacenamiento de grandes datos.

Con el fin de validar THS, nosotros creamos una colección de aproximadamente doce mil

tweets etiquetados en base a términos médicos. Estos tweets contienen uno o más términos médicos específicos y las etiquetas indicando si el tweet está relacionado con una condición médica o no. Nosotros usamos esta colección para probar varios modelos de aprendizaje automático, modelos basados en redes neuronales recurrentes y convolucionales. Nuestros experimentos muestran que nosotros podemos clasificar tweets con 96% de precisión, 91 % de recall y 86% de F1 Score. Estos resultados comparan favorablemente con las investigaciones recientes en esta área y muestran la promesa de nuestro sistema THS para identificar que mensajes están realmente relacionados con condiciones médicas.

Copyright © 2018

*by*

*Cristian Camilo Garzón Alfonso*

*DEDICATION*

*To my parents, María E. Alfonso Prada and José I. Garzón Camacho. To my brother,  
Wilmer E. Garzón Alfonso and my nieces, Sharon N. and Jael M. Garzón Rojas.*

## Acknowledgments

First and foremost, I would like to thank God for giving me the strength, knowledge, ability and opportunity to persevere and complete this degree satisfactorily. Next, I must thank my mother María Esther, my father José, and my brother Wilmer for their love, support, encouragement and patient through these years. Without them this achievement would not have been possible.

I would like to express my sincere gratitude to my advisor, Professor Manuel Rodríguez Martínez, for giving me the opportunity to work with him, for the continuous support of my Master of Science studies and related research and for his patience, motivation, teachings, friendship, and knowledge. His guidance helped me in all the aspects on the path of researching and writing this thesis.

I would also like to thank the Twitter Health Surveillance (THS) project team Danny Villanueva, Andres Hernández, and Luis Rivera for making this research possible and for assisting me in the completion of my degree and research. Last but not least, I would like to thank my friends in Colombia and Puerto Rico for the support, love, and always believing in me.

This research is supported by the US National Library of Medicine of the National Institutes of Health (NIH) under award number R15LM012275. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH. Some results presented in this thesis were obtained using the Chameleon testbed supported by the National Science Foundation (NSF).

# Contents

|                                   |             |
|-----------------------------------|-------------|
| <b>Abstract</b>                   | <b>ii</b>   |
| <b>Abstract (Spanish Version)</b> | <b>iv</b>   |
| <b>Acknowledgment</b>             | <b>viii</b> |
| <b>List of Figures</b>            | <b>xiv</b>  |
| <b>List of Tables</b>             | <b>xv</b>   |
| <b>1 Introduction</b>             | <b>1</b>    |
| 1.1 Motivation . . . . .          | 1           |
| 1.2 Objectives . . . . .          | 3           |
| 1.3 Contributions . . . . .       | 4           |
| 1.4 Outline . . . . .             | 5           |
| <b>2 Survey of the Literature</b> | <b>6</b>    |
| 2.1 Introduction . . . . .        | 6           |
| 2.2 Neural Networks . . . . .     | 6           |
| 2.3 Deep Learning . . . . .       | 9           |
| 2.4 Sentiment Analysis . . . . .  | 10          |

|          |  |           |
|----------|--|-----------|
| <b>3</b> | <b>Problem Statement</b>               | <b>13</b> |
| 3.1      | Description . . . . .                  | 13        |
| 3.2      | Formalization . . . . .                | 14        |
| 3.3      | Data Processing Pipeline . . . . .     | 15        |
| <b>4</b> | <b>System Architecture</b>             | <b>17</b> |
| 4.1      | Software Building Blocks . . . . .     | 17        |
| 4.2      | Big Data Framework . . . . .           | 18        |
| 4.3      | Data Storage . . . . .                 | 21        |
| 4.4      | Machine Learning Framework . . . . .   | 22        |
| 4.4.1    | Hyperparameters Matrix . . . . .       | 25        |
| 4.4.2    | Embedding Layer . . . . .              | 27        |
| 4.4.3    | Metrics . . . . .                      | 28        |
| 4.5      | Deep Learning Models . . . . .         | 29        |
| 4.5.1    | Recurrent Neural Networks . . . . .    | 29        |
| 4.5.2    | Convolutional Neural Network . . . . . | 32        |
| <b>5</b> | <b>Performance Evaluation</b>          | <b>35</b> |
| 5.1      | Hardware . . . . .                     | 35        |
| 5.2      | Software . . . . .                     | 36        |
| 5.3      | Experimental Results . . . . .         | 37        |
| 5.3.1    | Experimental Methods . . . . .         | 37        |
| 5.3.2    | Finding the Best Models . . . . .      | 39        |
| 5.3.3    | Final Results on RNN . . . . .         | 39        |
| 5.3.4    | Final Results on CNN . . . . .         | 42        |
| 5.3.5    | Discussion of Results . . . . .        | 43        |
| <b>6</b> | <b>Conclusion and Future Work</b>      | <b>45</b> |

|   |           |
|---|-----------|
| <b>References</b>                                 | <b>47</b> |
| <b>Appendices</b>                                 | <b>53</b> |
| <b>A Experimental Results for Classes 0 and 2</b> | <b>54</b> |
| A.1 RNN Results for Class 0 . . . . .             | 54        |
| A.2 RNN Results for Class 2 . . . . .             | 55        |
| A.3 CNN Results for Class 0 . . . . .             | 57        |
| A.4 CNN Results for Class 2 . . . . .             | 58        |
| <b>B GitHub Repositories</b>                      | <b>60</b> |
| B.1 Big Data Platform . . . . .                   | 60        |
| B.1.1 Machine Learning Platform . . . . .         | 60        |

## List of Abbreviations

**ADM** Advanced Data Management

**AI** Artificial Intelligence

**API** Application Program Interface

**CDC** Center for Disease Control

**CNN** Convolutional Neural Networks

**CSV** Comma Separated Values

**GPU** Graphic Processing Unit

**GRU** Gated Recurrent Units

**HDFS** Hadoop Distributed File System

**HHS** Health & Human Services

**HQL** Hive Query Language

**IT** Information Technology

**LSTM** Long Short-Term Memory

**ML** Machine Learning

**NIH** National Institutes of Health

**NLP** Natural Language Processing

**NSF** National Science Foundation

**RDD** Resilient Distributed Dataset

*CONTENTS*

*CONTENTS*

**ReLU** Rectified Linear Unit

**RNN** Recurrent Neural Networks

**SQL** Structured Query Language

**THS** Twitter Health Surveillance

**UPRM** University of Puerto Rico Mayagüez Campus

**US** United States

**WHO** World Health Organization

# List of Figures

|     |   |    |
|-----|---|----|
| 2.1 | Basic neural network architecture. . . . .                        | 7  |
| 2.2 | Complex neural network architecture. . . . .                      | 9  |
| 3.1 | Tweet processing pipeline. . . . .                                | 15 |
| 4.1 | Detailed process in THS big data architecture. . . . .            | 19 |
| 4.2 | THS database schema. . . . .                                      | 21 |
| 4.3 | THS machine learning system. . . . .                              | 23 |
| 4.4 | General Recurrent Neural Networks (RNN) architecture. . . . .     | 30 |
| 4.5 | Real example processed by RNN architecture. . . . .               | 31 |
| 4.6 | General Convolutional Neural Networks (CNN) architecture. . . . . | 33 |
| 4.7 | Inception CNN architecture. . . . .                               | 34 |
| 5.1 | THS cluster architecture. . . . .                                 | 35 |

# List of Tables

|      |   |    |
|------|---|----|
| 2.1  | Activation functions used in the Machine Learning (ML).                     | 8  |
| 4.1  | RNN hyperparameters.  | 26 |
| 4.2  | CNN hyperparameters.  | 27 |
| 4.3  | Confusion matrix to calculate the metrics.                                  | 28 |
| 4.4  | Scenario cases to experiment in RNN architecture.                           | 32 |
| 4.5  | Scenario cases to experiment in CNN architecture.                           | 34 |
| 5.1  | Hardware description by each component used in THS cluster.                 | 35 |
| 5.2  | Advanced Data Management (ADM) laboratory workstation hardware description. | 36 |
| 5.3  | Chameleon cloud nodes hardware description.                                 | 36 |
| 5.4  | Version for software packages used in THS.                                  | 37 |
| 5.5  | Distribution of tweets per class label.                                     | 37 |
| 5.6  | Class weight penalties.   | 38 |
| 5.7  | Precision results per RNN model.  | 40 |
| 5.8  | Recall results per RNN model.   | 40 |
| 5.9  | F1 Score results per RNN model.   | 41 |
| 5.10 | Execution time per RNN model.   | 41 |
| 5.11 | Precision results per CNN model.  | 42 |

|      |  |    |
|------|--|----|
| 5.12 | Recall results per CNN model. . . . .                | 42 |
| 5.13 | F1 Score results per CNN model. . . . .              | 43 |
| 5.14 | Execution time per CNN model. . . . .                | 43 |
| A.1  | Precision results per RNN model for class 0. . . . . | 54 |
| A.2  | Recall results per RNN model for class 0. . . . .    | 55 |
| A.3  | F1 Score results per RNN model for class 0. . . . .  | 55 |
| A.4  | Precision results per RNN model for class 2. . . . . | 56 |
| A.5  | Recall results per RNN model for class 2. . . . .    | 56 |
| A.6  | F1 Score results per RNN model for class 2. . . . .  | 57 |
| A.7  | Precision results per CNN model for class 0. . . . . | 57 |
| A.8  | Recall results per CNN model for class 0. . . . .    | 57 |
| A.9  | F1 Score results per CNN model for class 0. . . . .  | 58 |
| A.10 | Precision results per CNN model for class 2. . . . . | 58 |
| A.11 | Recall results per CNN model for class 2. . . . .    | 58 |
| A.12 | F1 Score results per CNN model for class 2. . . . .  | 59 |

# Chapter 1

## Introduction

### 1.1 Motivation

Public health officials, hospital directors, and other professionals related with health disciplines have to track and report disease outbreaks that affect populations around the world. Typically, the data comes in reports and CSV files from hospitals, and private doctor's offices. Typically, these reports are generated manually, increasing the risk of human error contained in transcript, analysis, charts, and different indicators that are used by professional organizations such as the United States (US) Center for Disease Control (CDC), World Health Organization (WHO), or the US Health Human Services (HHS). The processing and understanding of all these data might take weeks and the official warnings to a population could arrive too late. Poor and undeserved communities normally are highly affected since limited access to medical services often means that medical care attend the outbreaks when the major part of the community is already affected.

Social networks like Twitter, Facebook and Instagram provide a wealth of information about a lot of topics being discussed by all type of persons. Mining these conversations provides insights on topics such as sports, political activities, diseases, etc. These social networks are generating different type of interactions like replies, re-tweets, likes, comments, re-posts, etc. According to [1], here are some statistics of the activities in these networks:

- **Twitter** users post 656 million tweets per day
- **Facebook** users like 5.75 billion posts per day
- **Instagram** users post 67 million posts per day

All of this information is stored in databases and the amount of information is so large that it is difficult to inspect each message manually to review the content. A percentage of this information is public. Therefore, developers need to access these data via an Application Program Interface (API) of each social network.

At the University of Puerto Rico, Mayagez Campus we developed the Twitter Health Surveillance System THS for monitoring the Twitter social network in search for clues about the diseases mentioned. For THS it is important to collect and store the data in our own big data infrastructure. This makes it possible to process all information easily and fast, getting the data ready to be analyzed with Machine Learning (ML) algorithms. These algorithms can provide: a) information as to whether a message is about a disease or not, b) trending topics, and c) statistics about user engagements in a given geographical region.

In this project, we developed the infrastructure necessary to feed the tweets into a machine learning pipeline that can efficiently learn to classify the tweets as being related or not with diseases, and then uses those models in a production environment. All of this information can be provided to public health officials through a web-based dashboard. For the ML classification process, it is necessary to have a specific infrastructure to help researchers mine and identify the specific data necessary to the algorithm. The input of the ML algorithm is a training set that contains data examples with a label for each one, thus allowing for the algorithm to learn about what are the type of inputs that it will receive and what should be the output to predict. The ML algorithm ran on a Graphic Processing Unit (GPU), which runs instructions in parallel for better performance. However, our algorithms can also be run on regular CPUs. Our project provides the software tools

to help train the models, using neural networks, and then use these in a program that captures live tweets from the Twitter API.

## 1.2 Objectives

- **To investigate and implement an infrastructure to capture a live stream of tweets:** The streaming process includes operations to get, filter, and clean the data from the Twitter Streaming API, and store the processed tweets into the THS warehouse. For THS, it is important to save the record and all the tracking of the tweets, which will ensure the correct management, integrity and veracity of the tweets.
- **To investigate and implement machine learning algorithms for classification in THS:** The input necessary to the ML algorithm are the statuses collected from Twitter API streaming. For THS, we used a supervised algorithm to train, therefore the training set of tweets is labeled by hand. Thus, a group of people was required to complete this task by reading each tweet in a training set, and then classifying each tweet into a specific classification class. The output of the ML algorithm is a label classifying the input tweet in one of the three classes: a) the tweet talks about a disease, b) the tweets do not talk about a disease, or c) the tweet is ambiguous.
- **To test the infrastructure and the algorithms with big data tools:** Big data collecting and ML infrastructure need to be tested. Data collecting components were deployed, and tested at the THS's cluster that is located in the Electrical and Computer Department of the University of Puerto Rico Mayagüez Campus (UPRM). ML infrastructure was implemented, configured, and tested in a physical computer in the ADM laboratory at UPRM. Some tests for this part also were run in the NSF Chameleon cloud environment with better GPU resources than the physical

machine.

## 1.3 Contributions

This thesis represents original contributions by this author to the problem of mining social network data to detect conversations related with medical conditions, and help build decision support systems for medical applications. In particular, our contributions are as follows:

- **Describe how the problem of searching and mining for diseases on social media can be formulated as classification problem:** Several social media APIs are available to interact with the developers interested in using the public data that each API has to offer. For the THS project the public data comes from the Twitter Streaming API. In THS, we use several logic filters to select tweets that are related with the diseases of interest. These filters are applied during the data capturing process, and they check for the occurrence of certain keywords (e.g., flu). This facilitates the classification and cleaning by eliminating tedious manual work.
- **Present THS as a reference architecture for applications that need to process stream data with non-trivial methods:** THS is built with open source tools that implement streaming, deep learning, and data warehousing. By using open source tools, we can minimize the cost and increase the size of the developer pool for this type of solutions. This can make this type of solution more affordable to small and medium companies. This is important since the cost in the Information Technology (IT) area for medical informatics companies is a decisive factor to accomplish their goals.
- **Present a series of deep learning models that can be used to determine if a tweet is related with a disease or not:** We used deep learning methods, specifically RNN and CNN to create models that can classify the tweets into three

classes: class 0 - not related with a disease, class 1 - related with a disease, and class 2 - cannot be determined.

- **Describe an evaluation of these models on a real data set extracted from the Twitter Streaming API and identify best models for specific metrics:**

For RNN and CNN, we tested a set of hyperparameters to evaluate different models. We tested, all of these models, and took several measurements to evaluate which are the best. The best model has the highest value for these three metrics: a) Precision, b) Recall, and c) F1 score.

## 1.4 Outline

The outline of this thesis is as follows. Chapter 2 contains the survey of literature related with the machine learning, some techniques to use and a brief contextualization to facilitates the understanding of the THS solution. The problem description and the proposed solution is presented in Chapter 3. In Chapter 4, the system architecture for the big data and machine learning framework are shown in detail. It also describes the hyperparameters tuning, the embedding layer used in the machine learning architecture, the metrics used to measure our models, and the different models evaluated for RNN and CNN. Chapter 5 contains the performance evaluation of the system, as well as the description of the hardware and software environments used. Finally, the conclusions and future work is presented in Chapter 6.

# Chapter 2

## Survey of the Literature

### 2.1 Introduction

The analysis and processing of texts has captivated the attention of a large range of computing fields, especially Artificial Intelligence (AI), where it has been consolidated as a discipline known as Natural Language Processing (NLP). The work in [2] defines NLP as *“theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications”*. Thus, NLP can be defined the ability to infer, gain insight and act accordingly from the core representation of knowledge of the humankind. This makes this field of study a continuous source of achievements and opportunities yet to be discovered.

Several algorithmic techniques have been applied and enhanced iteratively to achieve a better understanding of natural language and texts. These include decision trees, hidden Markov models, supervised machine learning approaches, statistical processing, and deep learning techniques, most of which have turned out to be very successful.

### 2.2 Neural Networks

A neural network is a collection of neuron-like computational units. These are connected and organized under a user-defined network topology. Their goal is to perform distributed computations aimed to achieve a certain goal (e.g., classify an image as being that of a cat). Neural networks are modeled to resemble the inner workings of human brain.

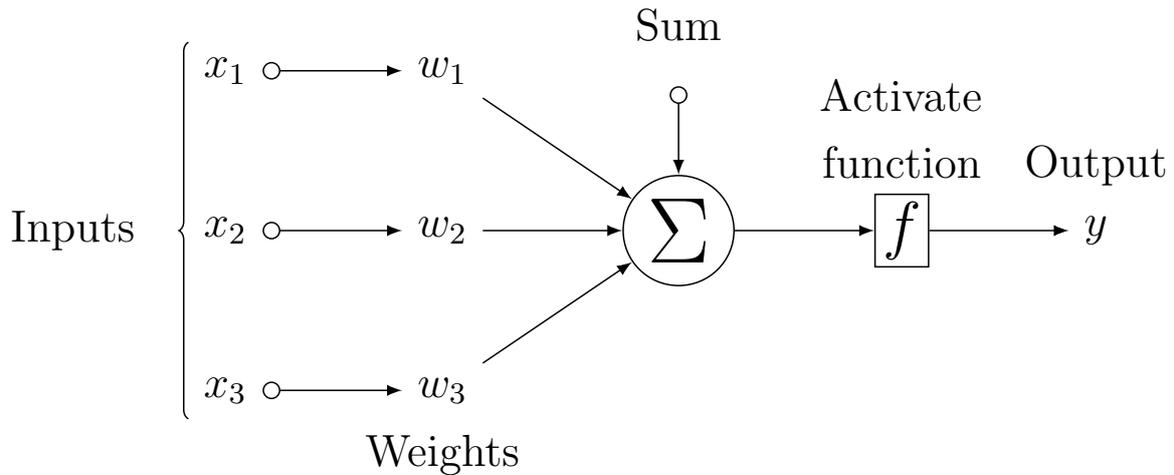


Fig 2.1: Basic neural network architecture.

Figure 2.1 shows a basic, one-neuron neural network, where the inputs comprise the left-side layer and output comprise right-side layer. The neuron takes the inputs  $\{x_1, x_2, x_3\}$ , and combines them with a summation operation. Each term  $x_i$  gets multiplied by a weight  $w_i$  to specify its importance. The summation is then passed to an activation function  $f$ , that outputs the class to which the input represented by  $\{x_1, x_2, x_3\}$  belongs. Weights are specific and selected for each neural network independently. These weights are necessary to minimize the cost of activation function generating the output value, which is the prediction from the algorithm. The work in [3] provides a mathematical model of a neuron as a system composed of an input function, an activation function and an output. The following equation represents the output activation:

$$a_j = g\left(\sum_{i=0}^n w_{ij} a_i\right), \quad (2.1)$$

where  $a_i$  is the output activation of unit  $i$ ,  $w_{ij}$  is the weight from link  $i$  and  $g$  is the activation function that determines the output. Typically,  $g$  is a sigmoid function [4], although other alternatives are the Hyperbolic Tangent function better known as TanH [5], Rectified Linear Unit (ReLU) function [6], or Normalized Exponential Function better known as Softmax Function [7]. Table 2.1 shows the graphical plot and formula for each of the activation functions mentioned before.

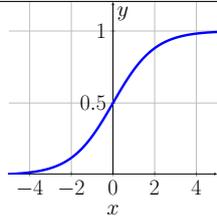
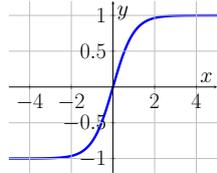
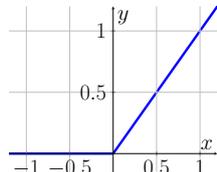
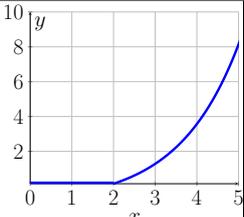
| Name    | Plot  | Equation   |
|---------|---|--|
| Sigmoid |    | $f(x) = \frac{1}{1 + e^{-x}}$  |
| TanH    |    | $f(x) = \frac{2}{1 + e^{-2x}} - 1$   |
| ReLU    |    | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Softmax |  | $f(x) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$  |

Table 2.1: Activation functions used in the ML.

Figure 2.2 shows a multilayer forward network where the hidden layer receives inputs from the previous layer. In this type of neural networks the outputs of nodes in one layer are the inputs to the nodes in the next layer. All of these inputs are combined using a weighted linear combination [8]. The result is calculated with an activation function, and depending on the problem to be solved, the function could use linear or nonlinear activation to produce final output.

Within the NLP context, neural networks have been used widely as part of the connectionist approach to NLP [2], where models are networks of simple units connected through links between layers. Two kinds of models are differentiated in this approach: 1) localist - where each unit represents a word concept, and 2) distributed - where multiple activation of units represent a concept.

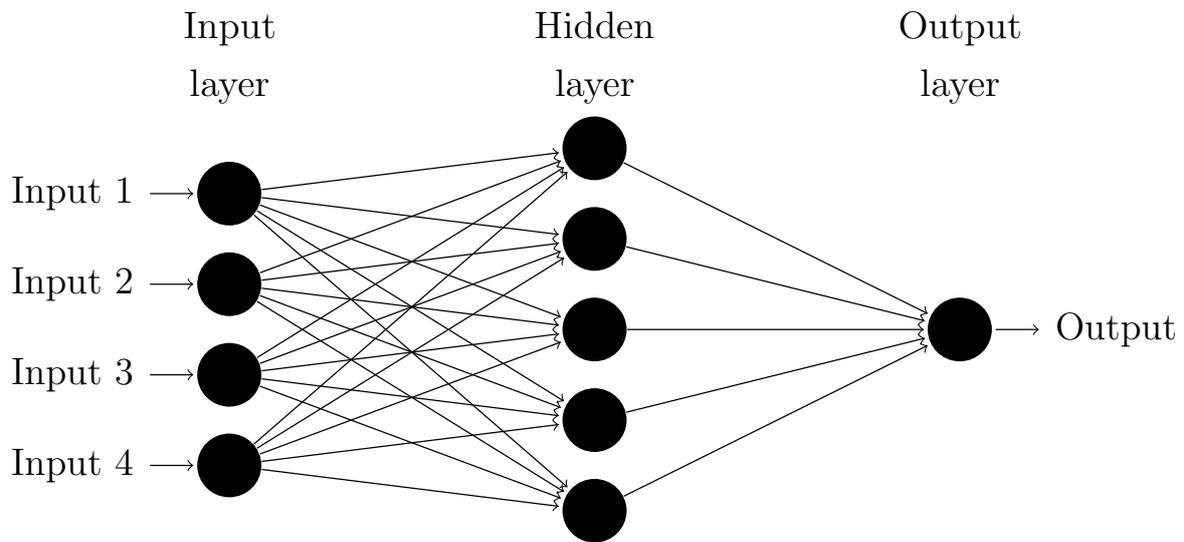


Fig 2.2: Complex neural network architecture.

## 2.3 Deep Learning

Deep learning has gained traction given its impressive results in several fields of artificial intelligence previously dominated by other AI techniques [9]. In simple terms, [10] defines deep learning as “machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts”. There have been several applications of deep learning techniques in the recognition and processing of texts, including:

1. The use of CNN to recognize words in images with unconstrained characteristics like words with unknown length, using multi-task learning and training with generated data an achieving state-of-the-art accuracy [11].
2. Integration with computing vision approaches to read digits from photographs using unsupervised feature learning methods [12] and deep CNN. These methods are used to localize, segment, and recognize patterns. These latter methods stabilize the model performance as the number of layers in the model is increased [13]. In some cases, the training data is generated artificially. This reduces the data acquisition cost and the deep neural network model take advantage over the others used for text recognition in images as a whole [14].
3. Speech recognition systems built for end-to-end deep learning that do not require intermediate data representations, and handle noisy environments better than other systems [15].

The examples in the previous paragraph denoted NLP tasks under unimodal scenarios, in the sense that a single form of data representation was considered. The work in [16] introduced a novel approach for multimodal learning of features over multiple data representations like video, audio and text, using deep networks. This approach demonstrated the use of cross modality feature learning to enhance the learning over one modality if there is data available of other data modalities.

## 2.4 Sentiment Analysis

Sentiment analysis is the field of study that analyses the opinions, sentiments, appreciations, attitudes, and emotion of the persons who write a given text, message, review, tweet, or post on a website or social network. The term *sentiment analysis* first appeared in [17], this technique consists in identifying how the sentiments are expressed by the writer in texts where the words express an evaluation, or emotional state about the given topic. Mostly, these elements are related with products, services, organizations, political persons, and diseases. Although, in industry the term *sentiment analysis* is used more frequently, in academic forums it is known as *opinion mining*. According to [18] this term could be referenced by other names like *opinion extraction*, *sentiment mining*, *subjectivity analysis*, *affect analysis*, *emotion analysis*, and *review mining*.

When sentiment analysis is performed on a set tweets related with one or more diseases, many emotional states can be found. These include happiness, sadness, irony, sarcasm, revenge, among others. The data set used to train models that detect sentiment analysis must contain a diverse set of examples that cover many of opinions that encode these emotional states. Often, opinions are subjective, and everybody uses the language in different forms and expresses their thoughts in different ways. Hence the true meaning of sentence or phrase contained in a text depend heavily on the other sentences and phrases that form the context of the written text.

**Example 1** *Comparison between two movie reviews.*

“ Yeah, the movie was pretty good... :D ”

“ Yeah, the movie was pretty good... ( o\_o) ”

*The movie reviews cited before use the same words, but a particular component change, the emoji. Hence, the sentiment/opinion of both texts changes completely by just changing this symbol. The first line means that the client was happy watching the movie. In contrast, the second line means that the client was unsatisfied and was expecting more from the movie.*

Applying sentiment analysis to tweets has been explored in [17, 18, 19] as a method to detect the mood of people toward some disease, or to detect if there is a unusual rate of chat about a given disease. According to [18], the tweet’s text is a component relatively easy to analyze, because the length is composed by only 280 characters. Therefore, the authors are going straight to the point of the topic.

In some cases the emojis are used to help in the prediction process [20, 21, 22, 23]. One common method to extract the meaning of an emoji is to convert it to words. For example, the emoji “:)” is converted to: “happy face”. Hence, this emoji is used to convey happiness or comfort with a given situation. In some scenarios, might also denote sarcasm. Thus, this conversion process makes it easier for the algorithm to analyze the text and find the sentiment/opinion in a fast and efficient manner. An example of this scenario follows:

**Example 2** *Tweet related to a health condition.*

“ I couldn’t go to work, because I got diarrhea this morning :( ”

*In this instance, the message is clearly related with the diarrhea medical condition. This tweet conveys a feeling of sadness because the effects of the diarrhea will prevent this person to show up to work.*

In contrast, the following tweet uses the same words to express disappointment, and it is not truly related to a medical condition affecting the person.

**Example 3** *Tweet not related to a medical condition.*

“ My husband’s verbal diarrhea against the neighborhood shows that he is not happy. ”

*In this case the term diarrhea is being used to discredit the husband attitude, implying that the conversation of the person’s husband is improper, rude, and disrespectful towards the neighborhood.*

Sentiment analysis is being used in many fields. For example, the work in [19] describes an application to predict the sentiment of the tweets written by people in the U.S. during the 2012 presidential campaign. Their goal was to gauge opinions on the leading candidates to several elective offices. The developers of the application had to expand their dictionary with words that captures idiosyncrasies particular to the U.S. because the language normally used in tweets is informal and vernacular. This was a critical element since the use of these informal (“slang”) words adds a lot of noise to the tweet’s text and hinders

the sentiment classification process. Specifically, in political and health topics people tends to write sarcastic messages, and this adds a significant challenge to the algorithms. The sentiment analysis is used in THS to determine if the tweet is related with a medical condition or not. Is important to note that the mere occurrence of a filtered word in the tweet text does not guarantee that the tweet is related with the disease.

In fact, recent research work has focused on using Twitter as a tool to help uncover health trends i.e. in this previous work [24] they extracted the tweets from a large comprehensive corpus of tweets. Then, their methods found the frequent word sets and start to filter the Wikipedia's articles by queries. They started to note that the changes in the frequent term sets from Twitter and in the medically-related articles were affected by the public health conditions which was affecting some populations around the world. In this research [25] they used 5,000 hand-labeled tweets to train their ML model. Those tweets contained the word "influenza". They tested the Support Vector Machine (SVM) model obtaining 97% of correlation ratio between the onset of flu outbreak and the increase of news related with the flu. They predicted the first season on time, but for the second season was predicted 2 peaks out of time, because there were a lot of news in Twitter related with the swine flu. Those all non-related tweets added noise to the SVM model.

There have been numerous other works that have attempted to use Twitter to detect diseases from social interactions [26, 27, 28, 29, 30]. However, many of these approaches use keyword searches to collect social messages, and assign them to a particular disease class, and then begin the analysis. For example, a previously collected data set is processed in search for tweets that contain the word flu. Those messages can be further analyzed to predict the sentiment ("mood") of the message. Unfortunately, keyword-based methods can produce inaccurate results since the mere occurrence of a keyword *does not* necessarily means that the message is indeed related with a medical condition. Thus, an analysis based of this approach can mislead public officials into thinking that some medical condition is affecting a community because the keyword is trending in the social network for a given region. Keyword search can be used to find candidate tweets, but there must be a another step to determine if the tweet is relevant or not.

# Chapter 3

## Problem Statement

### 3.1 Description

The fundamental problem that we want to tackle is the ability to capture tweets from the live Twitter data stream, search each tweet looking for target medical keyword(s), and then determine if the tweet is actually related with an actual medical condition. As we mentioned before in Section 2.4, the mere occurrence of a keyword does not make the tweet related with an actual medical condition. The following tweets, contained in the labelled dataset captured with THS, show why this problem is not trivial.

**Example 4** *Tweet related to a health condition.*

My weekend is ruined because of my flu :(

*In this instance, the message is clearly related with the flu medical condition. This tweet conveys a feeling of sadness and disappointment because the effects of the disease will prevent this person from participating in planned activities for the weekend.*

In contrast, the following tweet is not related to a medical condition at all, but rather uses a disease to emphasize the rejection of a view.

**Example 5** *Tweet not related to a medical condition.*

That reporter's verbal diarrhea against the president shows she ain't fair.

*In this case the term diarrhea is being used to discredit a news report from a journalist, implying that the views expressed in her reporting are excessive, lengthy, and biased against the President. This type of tweets can be observed more frequently in the live Twitter stream*

when some controversy surrounding the president erupts. Thus, it would be a mistake to conclude that some stomach virus, or other medical condition associated with diarrhea is on the rise.

Sometimes, the content of a tweet is ambiguous, and it is hard to classify it as been related to a medical condition.

**Example 6** *Tweet that is hard to classify.*

Well, well the flu can help me skip the family reunion. #sad #happy.

*This example conveys a contradictory message. On one hand, it can be interpreted as a sign of relief by the author, feeling good that she/he will not need to attend a family reunion because of the flu. On the other hand, the person is actually telling us she/he has the flu, or is hoping to get it. Given the miserable symptoms of the flu, it is hard to image how can anyone celebrate getting sick in order to avoid a family reunion.*

## 3.2 Formalization

Our goal is collect a stream of tweets  $T = \{t_1, t_2, \dots, t_n\}$  and classify each tweet into one of three classes:

- 0 - does not pertain to a medical condition
- 1 - does pertain to a medical condition
- 2 - is ambiguous

This is a *supervised classification* problem with three target classes. To solve it, we must first select the target keywords that might be associated with the medical conditions of interest. Otherwise, we would need to test any tweet whatsoever, and that makes the classification problem very hard.

Let  $M = \{m_1, m_2, \dots, m_k\}$  be a collection of  $k$  medical terms. These medical terms represent some medical topic or condition of interest. For example, we might use terms like *flu*, *runny nose*, or *influenza*, all of which can be associated with the topic of *flu*. We want to filter the stream  $T$ , discarding tweets that do not contain keywords in  $M$ . Let us call the output of this filtering process  $T'$ .

We can now apply a classifier  $\hat{y}$  to each tweet  $t' \in T'$  to classify each one into one of our three classes. The classifier  $\hat{y}$  must first be trained on a sample of  $T'$ , and then it can

be used from that point on, as long the input tweets come from the same distribution. Thus, we must always pass tweets to  $\hat{y}$  that have been filtered with  $M$ . If we change  $M$ , we must retrain  $\hat{y}$  with a new training set that contains examples with the keywords now present in  $M$ .

In practice, the distribution of classes for our problem is not uniform, with class 1 being the majority, then class 0 comes next, and class 2 is a distant third. Hence, our classifier must deal with the *class imbalance* problem [31, 32]. We tackled this issue by using the penalty technique whereby, during training time, a large penalty is given to the classifier whenever it misclassifies an example that belongs to one of the minority classes. This prevents the classifier from always assigning examples to the majority class since the cost would not be minimized.

### 3.3 Data Processing Pipeline

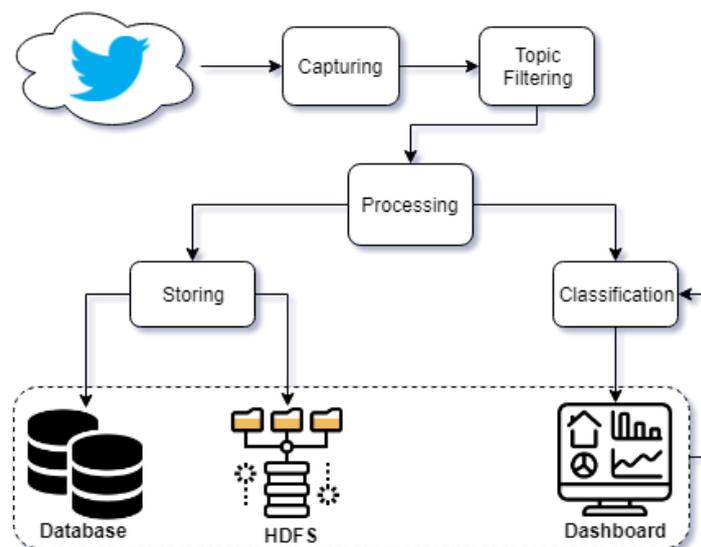


Fig 3.1: Tweet processing pipeline.

Conceptually, the tweets are processed using a pipeline as shown in Figure 3.1. As tweets are generated, they are captured and then filtered based on keywords so they can be assigned to a given topic. This capturing process can occur in two ways. One option is to subscribe to a sample of live tweets from the Twitter Streaming API. In this case, keyword search must be done after acquisition. The other option is to subscribe to a filtered stream, where a set of keywords, users, and locations can be specified

to narrow down the tweets of interest. Further filtering can be applied after data acquisition.

Next, tweets are routed for processing at additional stages. One stage performs classification as described in the preceding section. More stages can be added to perform custom-processing such as additional keyword filtering, emoji analysis, sentiment analysis, or computing target keyword frequency. The output from all these stages can be configured to go into dashboards, databases, HDFS, etc. In our case, raw tweets are always stored into HDFS to enable further analysis in the future. Notice that it is possible to re-ingest some of the output back into the classification or custom processing stages, perhaps to fine tune the results as models get re-calibrated. This can be used to continuously improve and adapt the model as new examples arrive over time.

# Chapter 4

## System Architecture

THS is a collection of daemons and web services that work together to collect, index, analyze, support queries on the tweets, and help to make predictions. THS is built atop: a) the Hadoop ecosystem of big data tools, and b) Keras, Google's Tensorflow, and Scikit-learn as ML tools. In this section we describe the various components in the system and their interactions.

### 4.1 Software Building Blocks

To design and develop THS, we used use different software tools that implement big data and machine learning capabilities. The following list describes these tools:

- **Python:** A programming language that provides support with all the tools we used: Twitter Streaming API, Kafka, Spark, Hive, Hadoop Distributed File System (HDFS), Yarn, Tensorflow, and Keras. All the code for this research was written in this language.
- **Twitter Streaming API:** An interface to get access in near realtime to public statuses which contain data such as: tweet text, user id, tweet id, language, etc. from random set of all the tweets.
- **Kafka:** A distributed streaming framework [33] which permits publish and subscribe to a queue where the records from the tweet stream are stored. These record are kept in Kafka for a while, for THS project this time is 3 hours.
- **Spark:** A unified analytics engine framework [34] which enables us to process, clean, group, and store all the tweets collected and read from the Kafka queue. Spark relies

on the Resilient Distributed Dataset (RDD) [35] abstraction to keep the tweets in the memory of the nodes in a computer cluster. RDD's run in-memory execution environment in parallel mode across the different nodes in the cluster.

- **Hive:** A data warehousing framework to enable reading, writing, and managing large datasets residing in distributed storage [36] using its own dialect of Structured Query Language (SQL) called Hive Query Language (HQL). All tweets are saved in structured tables that realize the hive database. HQL simplifies aggregation tasks, and permits us to communicate with Spark RDDs to extract huge amounts of data.
- **HDFS:** A distributed file system designed to run on commodity hardware [37]. HDFS runs in THS's cluster and provides high-throughput access to Hive tables data. HDFS is a highly fault-tolerant system. Hive tables are serialized and stored in HDFS [38], each table has a directory in HDFS.
- **YARN:** A resource management and job scheduling framework [39], which distributes and deploys the computational jobs into separate daemons across the nodes in the THS cluster. The master node of the cluster is identified in the network by the host named `masternode.ece.uprm.edu`. The master node is the ultimate authority that arbitrates resources among all the applications in the system.
- **Tensorflow:** A machine learning library which helps training the models, serving predictions, and refining the final results [40]. Tensorflow comes with built-in functionality to develop neural networks, clustering model, and other learning algorithms. It uses the python language, greatly simplifying its use to implement deep learning models. TensorFlow can be run in GPU and CPU, but it was designed to get the best performance in GPUs.
- **Keras:** A high-level machine learning API [41] which is written in python language, and capable of running atop Tensorflow. For the THS project, Keras will help with designing, training, testing, predicting, and refining the neural network models. Keras supports convolutional and recurrent networks.

## 4.2 Big Data Framework

The process necessary to collect, filter, and store the data from the Twitter API in the THS project is implemented with big data tools, mostly using Apache [42] open source projects. Figure 4.1 shows the tools used and the process detailed for the big data component.

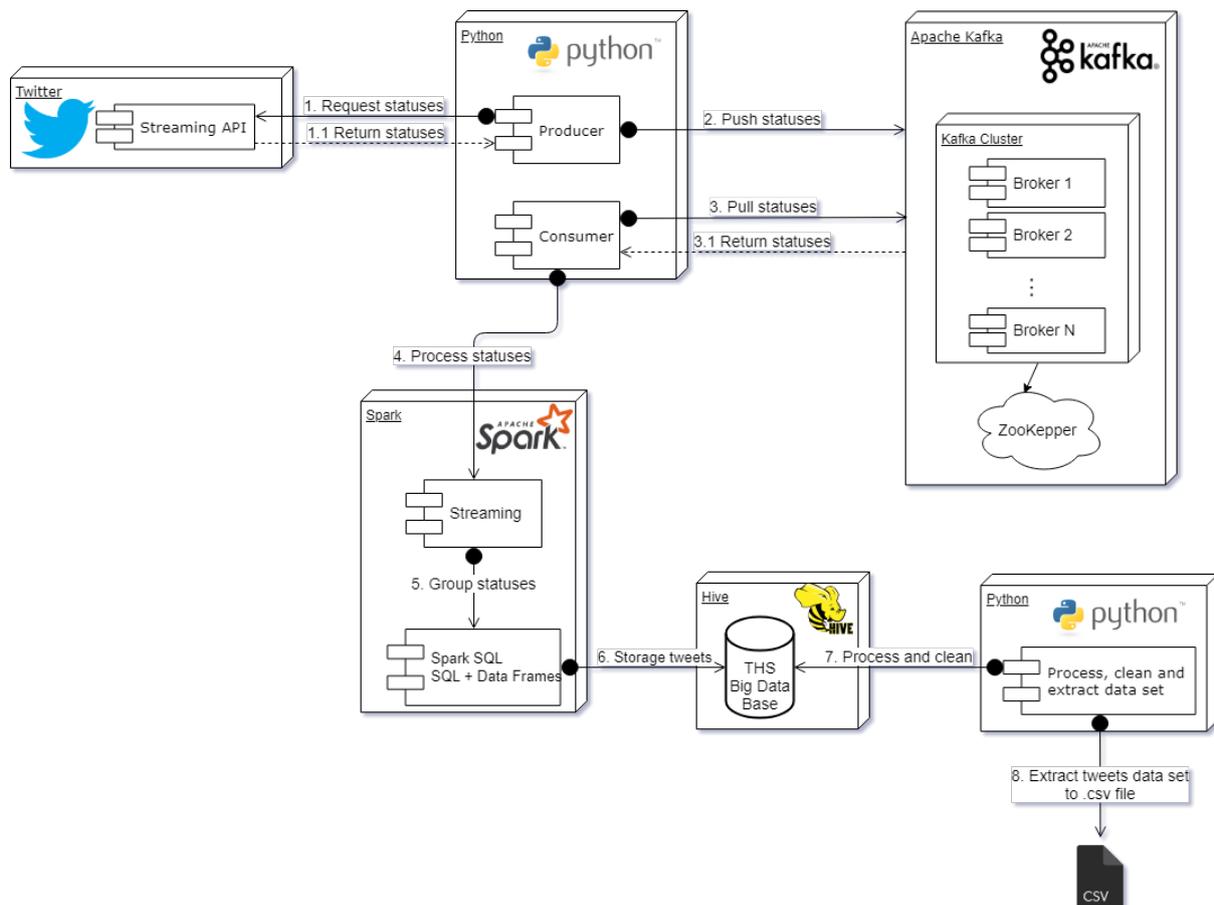


Fig 4.1: Detailed process in THS big data architecture.

The entire process, detailed in a step by step manner is as follows:

Step 1: A Python script called “Producer.py” is executed from the host node of the THS software system. In our case, we ran this script from a host in the UPRM Lockheed Martin Cloud, identified as node05.ece.uprm.edu. The script establishes a connection to the Twitter API through developer credentials, and starts collecting the tweets (known in the API as “statuses”). These statuses include original tweets, re-tweets, replies, likes, mentions, deletions, etc. Once captured, it is necessary to apply two filters to those statuses. Both of these filters are applied to the tweet’s text field. The first one is the language filter, which checks that the text is written in English. The second one is applied to the text, hashtags or mentions fields. This second filter looks for the words related with the diseases that are the interest to the user. If the text status contains one of the these words, the status (“tweet”) is automatically added to the warehouse. In our work, we used the following keywords for the diseases:

- (i) Zika
- (ii) Flu
- (iii) Ebola
- (iv) Measles
- (v) Diarrhea

It is important to note that the Twitter API just provides you with access to a random sample of about 1% of all the statuses being submitted[43].

Step 2: The python script puts the statuses collected in step 1 into a Kafka queue, from which they could be fetched for processing when ready. Thus, the queue act as a *buffer pool* that provides storage in RAM for the data. In any event, the statuses are buffered in the queue for 3 hours before Kafka automatically cleans the queue.

Step 3: In parallel, another python script, called “Consumer.py”, is running on the same node. This second script takes the statuses out of the Kafka queue, and feeds them into the Spark streaming system.

Step 4: Spark streaming starts to filter the data. In the THS project the important statuses are the original tweets, not the ones that are replies or re-tweets.

Step 5: Once the data has been filtered, a Spark *data frame* is created to manage the data and establish connection with the database in Hive. The data frame is a data structure used in Spark to group a collection (“batch”) of related tuples. In this part of the process for THS it is necessary to save the raw tweets into a table, as are received from the Twitter API. Another table is used to save some metadata information from the tweet, specifically, fields like: tweet ID, user ID and full text. The database schema is explained in the Section 4.3.

Step 6: The processed tweets are saved in the Hive data warehouse. This process is repeated in a loop in order to consume all tweets contained in a data frame. In our experiments, we collected and processed 56,013 tweets.

Step 7: Once the tweets are collected, it is necessary to define a sample of the data set to be used for training the neural network models that classify the tweets. For this, we another Python script. The first job of this script is to remove duplicate tweets, if they exist. Next, the script iterated over the text of all tweets and cleans each one

by removing special characters which are noisy to the machine learning algorithm, such as: ””, ””, ”-”, ”—”, etc.

Step 8: The final step is to export the tweet ID and the tweet text into CSV file. This file will be the input for the machine learning subsystem. The tweet id and the text are the only fields necessary for the machine learning algorithm.

### 4.3 Data Storage

As explained in the step 5 of the previous section, each tweet is mapped to records that are stored into tables of a database in the Hive data warehouse. Figure 4.2 shows the schema of this database. The tweet and raw\_tweet tables are filled up by the “Consumer.py” script as the automated data acquisition process. The other tables are filled up by manually running another script. The descriptions of these tables are as follows:

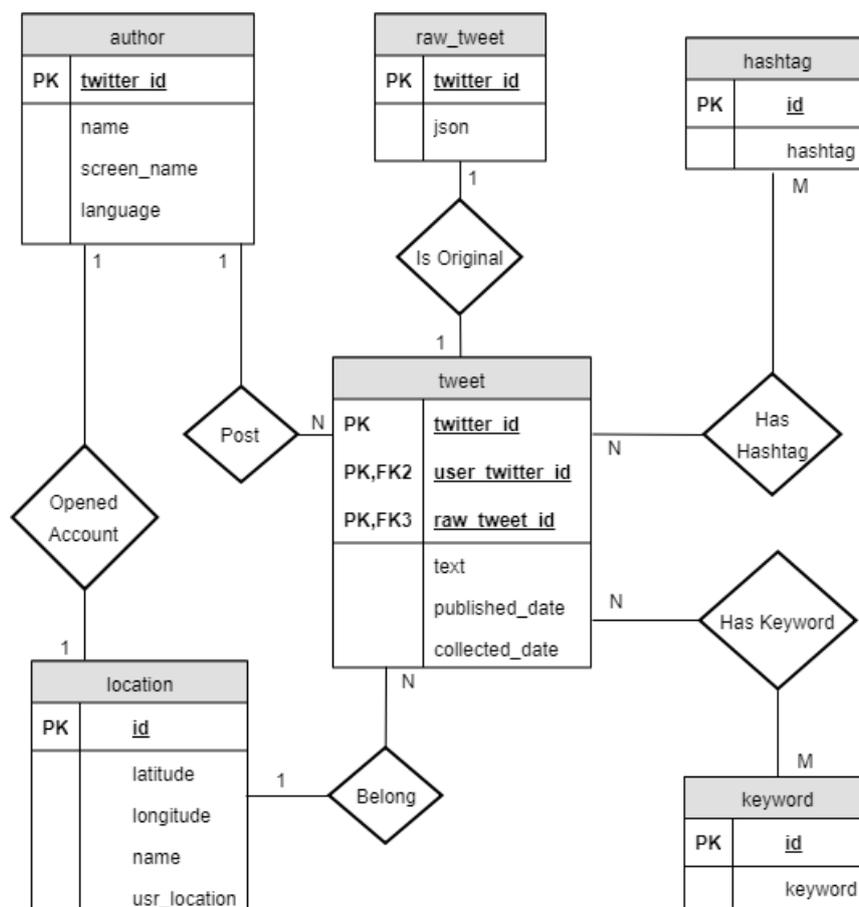


Fig 4.2: THS database schema.

- **Tweet:** Each time a tweet is captured, a new record is created in the tweet table that contains the following attributes: a) *twitter\_id* is the id assigned by Twitter to the tweet, b) *user\_twitter\_id* is the foreign key id that comes from the relation with author table, it contains the id of the author of the status, c) *raw\_tweet\_id* is the foreign key id that comes from the relation with raw\_tweet table, d) *text* is the tweet's full text (up to 280 characters) written by the author, e) *published\_date* is the date when the tweet was published, and f) *collected\_date* is the date when the tweet was collected on THS.
- **Raw tweet:** a record in this table contains a) status id assigned by Twitter b) the status as received from Twitter in the form of a JSON string.
- **Author:** if the author of a tweet is not already stored in the system, then we add a new author record. This record contains: a) the id of the author given by Twitter, b) the author's full name, c) the author's Twitter user name, d) the author's language of preference, and e) the location (if is provided) by the author (e.g., city, state, or country).
- **Hashtag:** a record for a hashtag entity is created, linking the tweet with each of the hashtags that is contains in the tweet text.
- **Keyword:** a record for a Keyword entity is created, linking the tweet with each of the target keywords that are contained in the tweet text.
- **Location:** a record to save the latitude, longitude, and name for the status (if is provided). Most users do not publish the location from where they wrote the tweet. Therefore, any search by location will try first to use this field, or by default use to the location of the author, which is the country where the account was opened.

## 4.4 Machine Learning Framework

The machine learning framework provides the tools to classify tweets into one of the target classes for disease relevance. The input to the machine learning framework is a CSV file that contains 2 columns, the first one is the tweet text, and the second one is the class label set manually during the labelling process. This file is generated from the data in the Hive warehouse. In our work, we created a training set composed of 12,500 tweets. This file was labelled manually by five members of the THS project team. The label class are:

- 0 - the tweets is not related with a disease.
- 1 - the tweets is related with a disease.
- 2 - ambiguous.

This file is then processed to transform the text into the tensor form expected by the Keras/Tensorflow tandem. The result can be presented in a dashboard, app, exported as a csv file, or stored back to some table in Hive.

Figure 4.3 shows the process and the tools used in each step. The entire process is detailed in a step by step manner as follows:

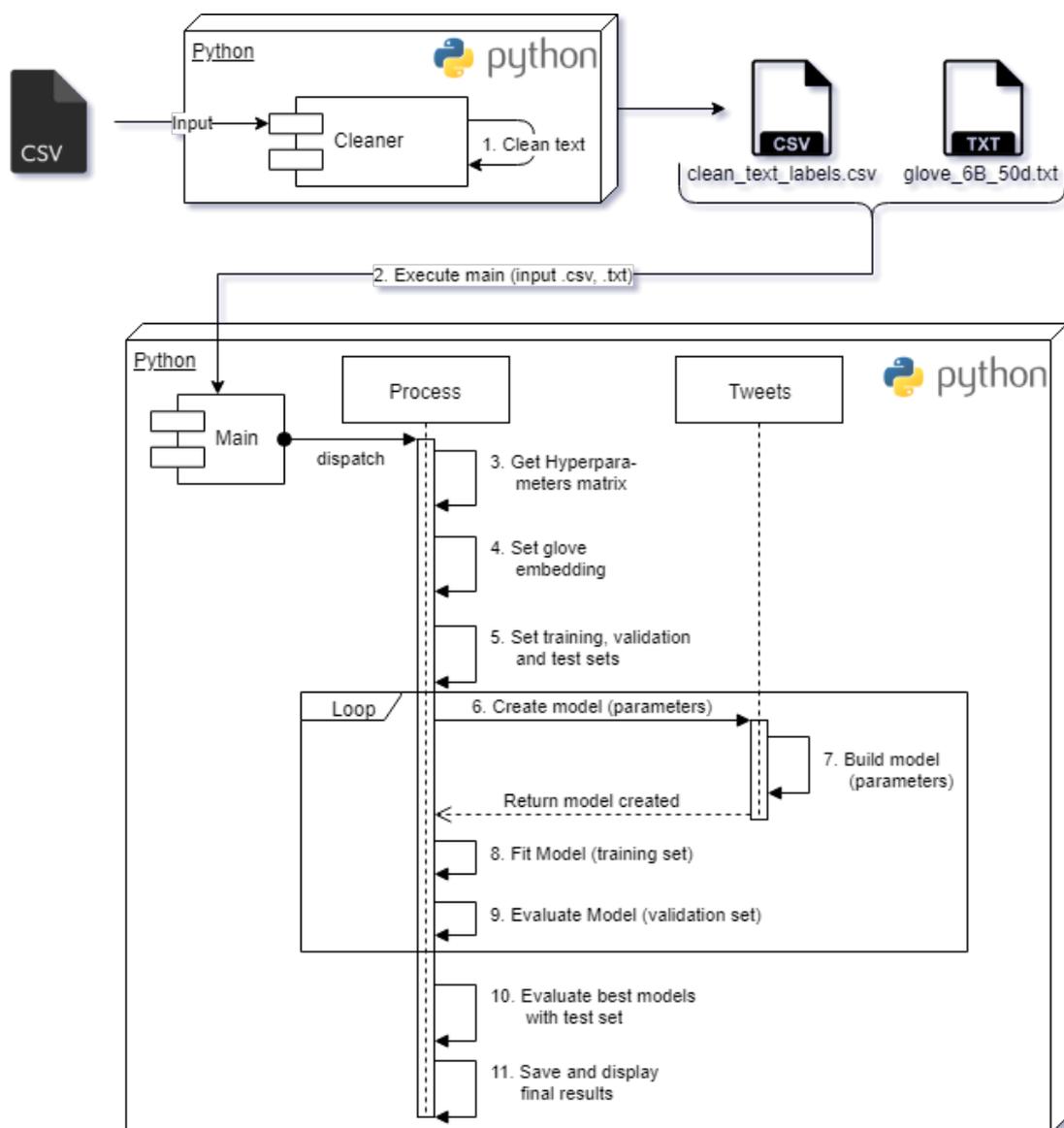


Fig 4.3: THS machine learning system.

- Step 1: A Python script called “Cleaner.py” is executed on a workstation. In our case, we used a workstation in either the ADM Lab or in a Chameleon cloud’s node. This script takes each line of the input file and removes the punctuation, emojis, web site links, and user mentions. In addition, it removes the “#” character from the hashtags. We do not use word stemming, since While developing the models and making initial training runs, no much difference was notice between using word stemming or not. The same applies to removing stop words or not. The script’s output is a CSV file with the tweet text for all the lines cleaned.
- Step 2: The machine learning script is executed in host node configured previously with Keras, and Tensorflow. We also run this script on a host that provides GPU capabilities. This script receives two files: 1) The cleaned csv file obtained from the previous step, and 2) A text file containing word embeddings. In our case, we used Stanford’s Glove word embedding, stored in a file called “Glove File.txt”, that contains a vector representation [44] for 400,000 words, called a *word embedding*. In particular, we used the embedding that contains vectors of fifty (50) dimensions trained on Wikipedia data. This is further discussed in step (4) below. The main class creates an instance of the process class, and passes the input name files as parameters.
- Step 3: A hyper-parameter matrix is defined with different values for some elements that will be part of the neural network. This matrix will permit the evaluation of different neural network models. The parameters to change could be number of epochs, learning rate, batch size, dropout, and some values that are part of the different layers. More details are presented in Section 4.4.1.
- Step 4: A glove embedding file enables a mapping of words to a vector in a  $n$ -dimensional space of real numbers. In our case the number of dimensions is 50. These vectors are used to convert each word of the tweet text in a numerical vector representation. This scheme has been shown to provide a better way to find features in the target text [45, 46]. More details are presented in Section 4.4.2.
- Step 5: The input CSV file contains all the text for the tweets with their respective labels. The data is split in three subsets: training, validation, and test. The first one is the training set that is created out of 60% of the total tweets. This set is used to train the neural network models. The second one is the validation set with 20% of all tweets. This second set is used to chose the combination of hyperparameters that provides the best performance with respect to a given metric (e.g., accuracy). The

last data subset is the test set containing the remaining 20% of the tweets. This last one is used to measure and report the final performance of the model.

- Step 6: As we can see from the Figure 4.3, the Process class create the Tweets class that contains the different neural network models to be trained and gives it the hyperparameters necessary for the model. Next, the computation becomes a loop (steps 6-9) with 1,120 or 1,080 iterations to evaluate RNN or CNN respectively. The entire process is explained in Section 4.4.1.
- Step 7: In each iteration, the Tweets class receives a different combination of hyperparameters, and based on these, creates and return a neural network model to the Process class.
- Step 8: Once the model is created and configured, the Process class starts to execute the fit function to train the model. For each record in the training dataset, the keras/Tensorflow algorithm starts to learn the label from the tweet text. This step is finished when the model is trained with all the records. In THS case are 7,500 the tweets to train the different models. This process is repeated  $n$  epochs. In our case, this was a hyperparameter.
- Step 9: Using the validation dataset and the model trained in the last step, the algorithm starts to read the tweet text and predict the label, comparing the prediction label with the real label value presents in the record. With this comparison, we calculate some metrics like precision, recall and F-1 Score. More detail is presented in Section 4.4.3.
- Step 10: In each iteration, the algorithm calculates the 5 best models and saves them in a Python dictionary. When the loop finishes the best models are tested with the test dataset to pick the best one.
- Step 11: The best model is used to start to predict more tweets related with diseases acquired from Twitter Streaming API.

#### 4.4.1 Hyperparameters Matrix

A hyperparameter is a parameter which cannot be directly learned or estimated from the training process. It is a predefined value set by the practitioner before the fitting and training process starts. It defines high-level properties of the model such as the learning rate, number of epochs to train, complexity, etc. Finding the best value for each

hyperparameter is not a trivial process. Initially, we used hyperparameter values based on results from previous researchers. Once, we had an idea about which were the most important values, we estimated additional values, and created a hyperparameter matrix to discover the combination that results in the best predictions. In this hyperparameter matrix, a row represents a different combination of parameters.

The training process is executed in a for loop which iterates through all the hyperparameters matrix's combinations. For each iteration a different row from the matrix was evaluated, and several performance metrics were calculated in the final steps to define the best five (5) and the worst three (3) models. The total number of combinations is given by the formula:

$$Total\ combinations = \prod_{i=1}^n V_i$$

where  $n$  is the number of hyperparameters options to define values, and  $V$  is the number of values assigned for each hyperparameter option.

## RNN Hyperparameters

The hyperparameters matrix defined to test the RNN model is shown in table 4.1.

| RNN use         | Hyperparameter | Values  |       |      |      |     |     |   |
|-----------------|----------------|---------|-------|------|------|-----|-----|---|
| Fit process     | Learning rate  | 0.001   | 0.003 | 0.01 | 0.03 | 0.1 | 0.3 | 1 |
|                 | Epochs         | 5       | 10    | 20   | 40   | 60  |     |   |
|                 | Batch size     | 32      |       |      |      |     |     |   |
| Compile process | Optimizer      | RMSprop |       |      |      |     |     |   |
| LSTM 1          | Layer units    | 50      |       |      |      |     |     |   |
| Dropout 1       | Rate           | 0       | 0.1   | 0.3  | 0.5  |     |     |   |
| LSTM 2          | Layer units    | 50      |       |      |      |     |     |   |
| Dropout 2       | Rate           | 0       | 0.1   | 0.3  | 0.5  |     |     |   |
| Dense Layer 1   | Layer units    | 32      | 64    |      |      |     |     |   |

Table 4.1: RNN hyperparameters.

The total number of combinations for RNN is 1120, this number was calculated from the formula:

$$Total\ combinations = \prod_{i=1}^9 V_i = 7 * 5 * 1 * 1 * 1 * 4 * 1 * 4 * 2 = 1120$$

## CNN Hyperparameters

The hyperparameters matrix defined to test the CNN model is showed in table 4.2.

| CNN use         | Hyperparameter | Values |          |         |       |     |
|-----------------|----------------|--------|----------|---------|-------|-----|
| Fit process     | Learning rate  | 0.001  | 0.003    | 0.006   | 0.008 | 0.1 |
|                 | Epochs         | 10     | 20       | 40      |       |     |
|                 | Batch size     | 32     |          |         |       |     |
| Compile process | Optimizer      | Adam   | Adadelta | RMSprop |       |     |
| Conv2D          | Filters        | 64     | 128      |         |       |     |
| Dropout         | Rate           | 0      | 0.1      | 0.3     | 0.5   |     |
| Dense Layer     | Layer units    | 128    | 256      | 512     |       |     |

Table 4.2: CNN hyperparameters.

The total number of combinations for CNN is 1080, this number was calculated from the formula:

$$Total\ combinations = \prod_{i=1}^7 V_i = 5 * 3 * 1 * 3 * 2 * 4 * 3 = 1080$$

### 4.4.2 Embedding Layer

Once the tweets have been pre-processed, we need to convert from text to a tensor representation. One option is to create a dictionary of words, with each word  $w_i$  having a position  $i$  in the dictionary. Then, a tweet can be represented with a one-hot encoding vector representation. In this scheme, a vector  $v$  representing a tweet  $t$  will have position  $v[i] = 1$ , if word  $i$  is present in the tweet, or 0 otherwise. However, this approach has two main drawbacks. First, since the dictionary can have thousands of words, the vector  $v$  can be very long and mostly contain 0s. Secondly, with one-hot representation the order of words within the tweet is lost and can yield inaccurate results. In THS, we use the well-known word embedding methodology [47], in which there is an embedding function that maps each word  $w_i$  in a tweet  $t$  into a vector  $v_i$  in an  $n$ -dimensional vector space  $R^n$ . A Tweet  $t$  then becomes represented as a  $m \times n$  matrix  $\mathcal{M}$ , where  $m$  is the longest tweet length and  $n$  is the dimension of the vector space. Conceptually, each row  $i$  in  $\mathcal{M}$  is a vector  $v_i$  representing word  $w_i$ . Since not all tweets have the same length, padding with one or more instances of a zero vector is need to make all tweets in a batch have the same length. In practice, the tweet  $t$  must first be mapped into a list of word indices  $L$ . Entry  $L[i]$  contains the position of word  $i$  in the dictionary used by the embedding. The

embedding takes this index  $L[i]$  and maps it to a vector  $v_i$  word embeddings provide a better representation of the data, and it has been shown that related words in the target language tend to be mapped to close vectors in the vector space [47]. Moreover, word embedding are amicable for processing by deep learning models based on RNN and CNN.

### 4.4.3 Metrics

Since the data set of labelled tweets is class imbalanced, we do not use accuracy as the evaluation metric. Instead we use precision, recall, and F1-score to evaluate each option. To calculate the metrics we used the following confusing matrix:

|                        | <b>Predicted: 1</b>  | <b>Predicted: 0</b>  |
|------------------------|----------------------|----------------------|
| <b>Actual Value: 1</b> | True Positive<br>TP  | False Positive<br>FP |
| <b>Actual Value: 0</b> | False Negative<br>FN | True Negative<br>TN  |

Table 4.3: Confusion matrix to calculate the metrics.

#### Precision

The *precision* metric measures the exactness of a classifier, in terms of how many examples of a class  $i$  it correctly classifies. Given a class  $i$ , the precision on class  $i$ ,  $P_i$  is defined as:

$$P_i = \frac{TP_i}{TP_i + FP_i}$$

Here  $TP_i$  is the number of correctly classified examples (true positive examples), while  $FP_i$  is the number of examples incorrectly labeled as belonging to class  $i$  (false positives). Thus, the precision on class  $i$  is a ratio between the number of correctly classified examples  $TP_i$ , and the sum of  $TP_i$  and  $FP_i$ . The closer  $P_i$  is to 1, the more exact the classifier is on class  $i$ .

#### Recall

The *recall* metric provides a measure of how complete is the classifier in correctly labeling the examples of a class  $i$ . Given a class  $i$ , the recall on class  $i$ ,  $R_i$  is defined as:

$$R_i = \frac{TP_i}{TP_i + FN_i}$$

As before,  $TP_i$  is the number of true positive examples for class  $i$ , whereas  $FN_i$  is the number of examples from class  $i$  that were *missed* by the classifier (false negatives). Recall is a ratio between  $TP_i$ , and the sum of  $TP_i$  and  $FN_i$ . In other words, recall tells what percentage of the examples of class  $i$  the classifier correctly detects and labels.

## F1 Score

Whether precision or recall is the right metric is a matter of debate (often a bitter debate). For some applications, recall is more important. The F1 score is a metric that seeks to balance precision and recall, proving a method to determine how balanced a classifier is. The F1 score for class  $i$  is defined as follow:

$$F1_i = 2 \frac{P_i R_i}{P_i + R_i}$$

The calculation of this metric is based on the *precision* and *recall* metrics.  $P_i$  is the precision metric for class  $i$  and  $R_i$  is the recall metric for class  $i$ . Notice that a classifier that is balanced will have an F1 score close to 1 since both the numerator and denominator will trend to 1. In contrast, a classifier biased toward either precision or recall will have a numerator that trends towards 0.

## 4.5 Deep Learning Models

THS uses RNN and CNN as the main ML building block for classification operations. RNN are designed for problems related with sequential data such as NLP. We used Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) for RNN, and inception architecture for CNN.

### 4.5.1 Recurrent Neural Networks

Figure 4.4 show the general architecture of the RNN that we used. This network is a classic encoder-decoder network. On the left, we have the batches of tweets to be fed into the network.

The first stage of the network is the embedding layer, which takes care of mapping each tweet  $t$  into a embedding  $\mathcal{M}$ . The embedding is then feed into the first recurrent layer, which can be configured to use either LSTM or GRU. This layer works as an encoder unit. Unless otherwise specified, the LSTM and GRU layers used as input a shape of 72

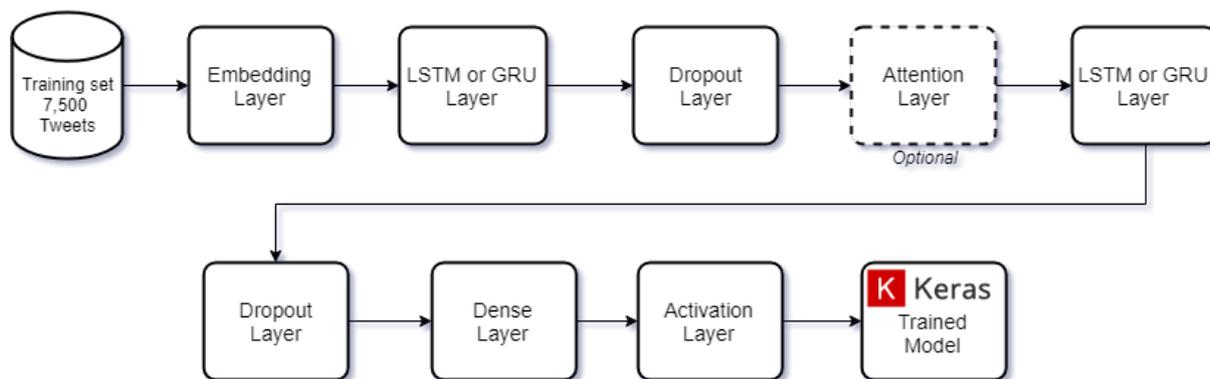


Fig 4.4: General RNN architecture.

units. This number comes from the maximum tweet length that our testing data sets had. The next layer is a dropout layer used to prevent overfitting on the first recurrent layer. An optional attention layer [48] is added next depending the model. This layer is used to help focus the RNN into sections of the tweets that might be more important than others. If the attention layer is used, then the RNN must output all intermediate sequence outputs. The next layer in the network is another recurrent layer that acts as the decoder component. Its output is passed to another dropout layer, and then to a dense layer containing 32 or 64 hidden units. The output of this layer is passed to a final softmax layer with 3 hidden unit which outputs a vector with the probabilities for each of the three classes.

For the following example, the Figure 4.5 shows the RNN architecture, and the structure to predict the label of this tweet. In this example we did not use the Attention Layer. The LSTM or GRU layers are connected between them. These connections between units enable the network to share the weights, and pass information on the important words read so far. This is the method by which memory is realized in the neural network.

**Example 7** *Tweet example for RNN Architecture.*

“ The flu is so dangerous, fatal and bad ”

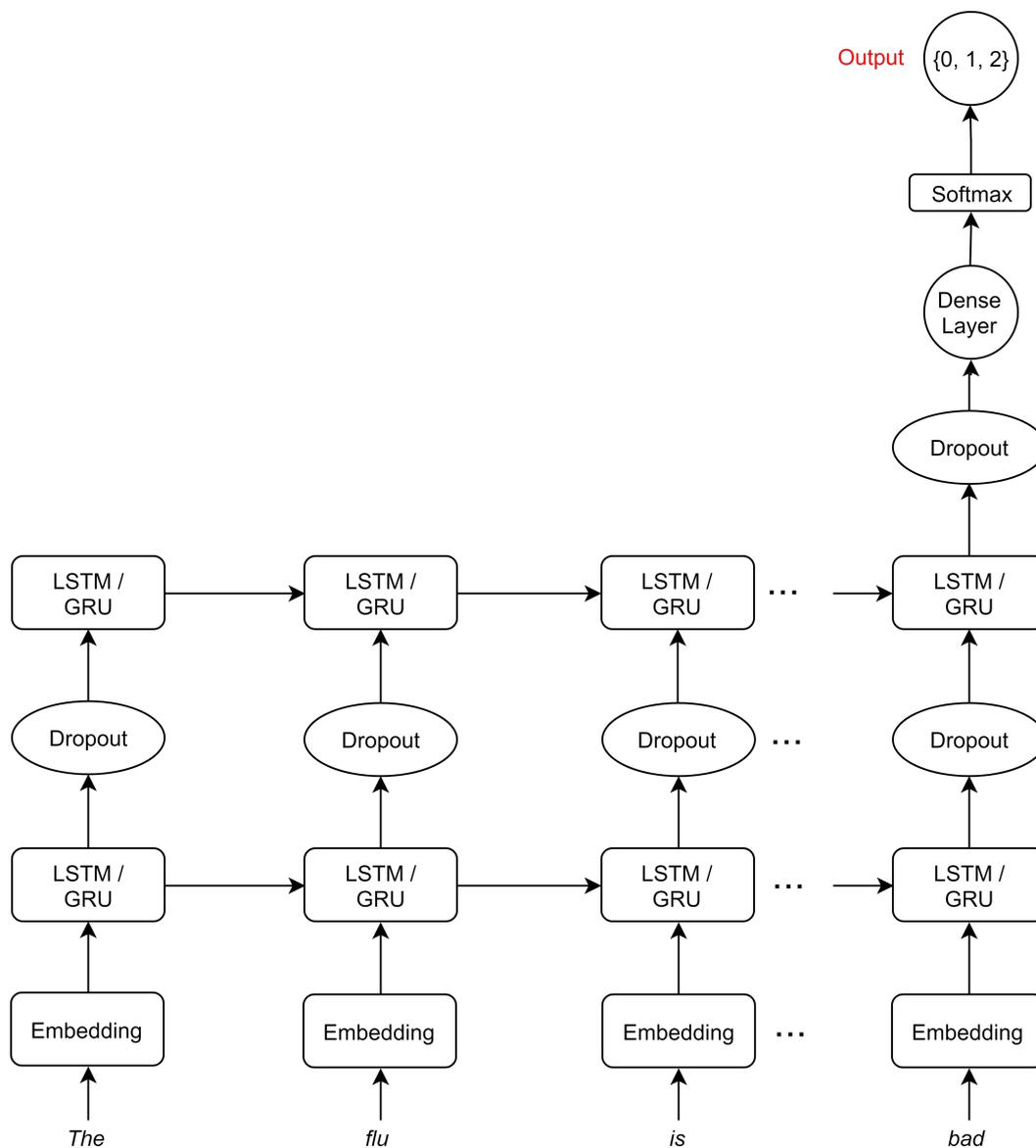


Fig 4.5: Real example processed by RNN architecture.

In our implementation, we provided the following concrete models based on this RNN architecture:

| Abbreviation                  | Description   |
|-------------------------------|---|
| 2 LSTM Attention 50 Units     | 2 LSTM layers with 50 hidden units each, and attention layer.     |
| 2 LSTM No Attention 50 Units  | 2 LSTM layers with 50 hidden units each, and attention layer.     |
| 2 LSTM Attention 100 Units    | 2 LSTM layers with 100 hidden units each, and attention layer.    |
| 2 LSTM No Attention 100 Units | 2 LSTM layers with 100 hidden units each, and no attention layer. |
| 2 GRU Attention               | 2 GRU layers with attention layer.                                |
| 2 GRU No Attention            | 2 GRU layers without attention layer.                             |
| LSTM GRU Attention            | LSTM layer followed by GRU layer with attention layer.            |
| LSTM GRU No Attention         | LSTM layer followed by GRU layer without attention layer.         |
| GRU LSTM Attention            | GRU layer followed by LSTM layer with attention layer.            |
| GRU LSTM No Attention         | GRU layer followed by LSTM layer without attention layer.         |

Table 4.4: Scenario cases to experiment in RNN architecture.

## 4.5.2 Convolutional Neural Network

Figure 4.6 shows the general architecture of the CNN that we used. On the left, we have the batches of tweets to be fed into the network.

The first stage of the network is the embedding layer, which takes care of mapping each tweet  $t$  into a embedding  $\mathcal{M}$ . The embedding is then feed into the first inception layer, where the kernel size dimension can be configured by a parameter  $n$ . We try  $n$  with values of 3 and 5. Additional optional inception layers could be added next depending the model to test. The *inception* technique was proposed by Google in [49, 50]. In our case we tried two scenarios, one with a single inception layer varying the kernel  $n$  value, and other scenario using the optional part which provides a complex model with four layers varying the kernel  $n$  value. The next layer is a flatten layer used to convert the convolutional matrix resulting from the inception layers to a one-dimensional vector. Next, we have a dropout layer used to prevent overfitting on the flattened vector. The last layer is a dense

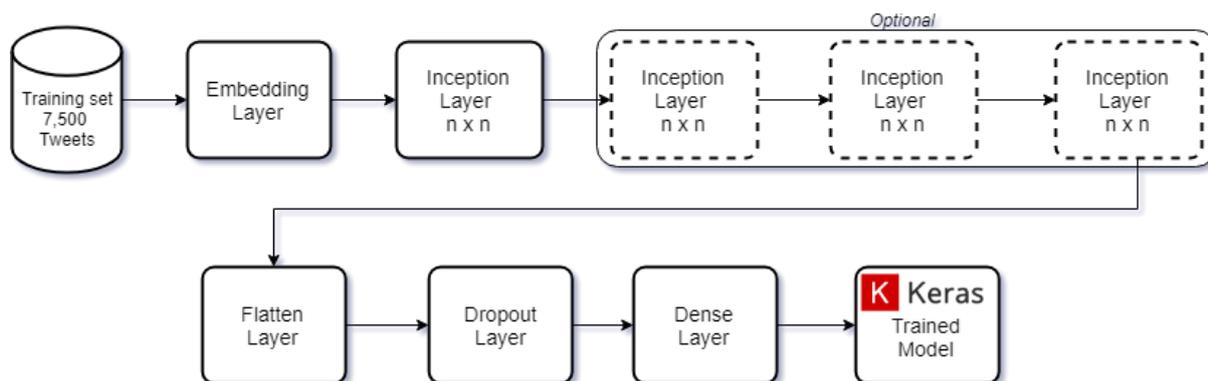


Fig 4.6: General CNN architecture.

layer containing 128, 256, or 512 hidden units. The output of this layer is passed to a final softmax layer with 3 hidden unit which outputs a vector with the probabilities for each of the three classes.

A problem with CNN is that by creating deep networks, in the hope of better performance, computational time became prohibitively large. In addition, problems like exploding gradients hurt the accuracy of the network. The main idea of the inception architecture is maximize the accuracy, and minimize the complexity, which enable savings in terms of training time. Is important to note that this model of inception was an evolution of other models previous tested. The work presented in [50] explains that one way to increment the performance in the neural networks is not modify the dimensions of the input drastically. A reduction in the input layers may cause loss of information, an issue known as *representational bottleneck*. A factorization in the convolutional kernel size was proposed, the  $n \times n$  convolutions could be represented in two convolutional layer one of  $n \times 1$  and the another of  $1 \times n$ . For example, a  $3 \times 3$  convolution is equivalent to perform a  $1 \times 3$  convolution, and then perform a  $3 \times 1$  convolution on its output. This factorization method is 33% more cheaper than the single  $3 \times 3$  convolution. The inception layer elements and architecture used for THS are shown in the Figure 4.7, where  $n$  was 3 or 5 respectively.

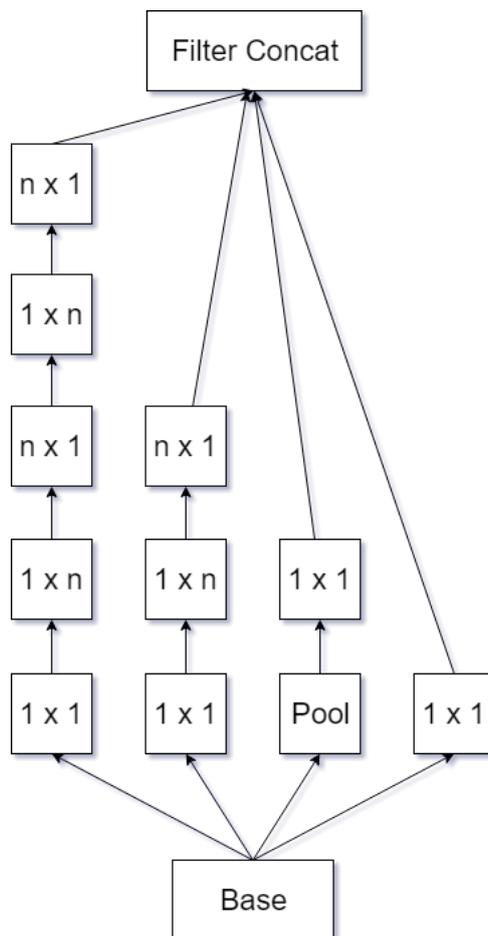


Fig 4.7: Inception CNN architecture.

In our implementation, we provide the following concrete models based on this CNN architecture:

| Abbreviation           | Description   |
|------------------------|---|
| Single inception 3 x 3 | 1 inception layer with kernel size of 3 X 3                 |
| Single inception 5 x 5 | 1 inception layer with kernel size of 5 X 5                 |
| 4 inceptions 3 x 3     | 4 interconnected inception layers with kernel size of 3 X 3 |
| 4 inceptions 5 x 5     | 4 interconnected inception layers with kernel size of 5 X 5 |

Table 4.5: Scenario cases to experiment in CNN architecture.

# Chapter 5

## Performance Evaluation

### 5.1 Hardware

To complete this research it was necessary to use different physical equipment. Figure 4.1 shows the first key hardware component which is a cluster composed of 12 physical hosts. It was installed and configured with the software presented in Section 5.2 to cover the big data architecture and requirements. These nodes ran Ubuntu 14.05 LTS on bare metal. Each host of this cluster has these specifications:

| Element    | Description                         |
|------------|-------------------------------------|
| Hard disk  | 297 GB                              |
| RAM Memory | 8 GB                                |
| Processor  | Intel(R) Xeon(R)<br>E3120 @ 3.16GHz |
| GPU        | None                                |

Table 5.1: Hardware description by each component used in THS cluster.

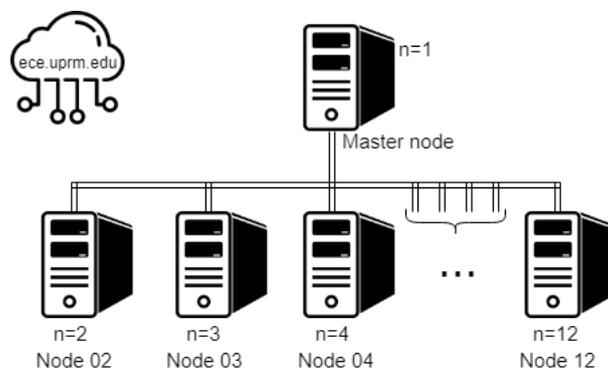


Fig 5.1: THS cluster architecture.

The second key resource, is a physical desktop computer located in ADM laboratory with the environment configured to code and test all the different models and developments needed to THS. The machine ran the Ubuntu 16.04 LTS OS. The hardware specifications of this computer are:

| <b>Element</b> | <b>Description</b> |
|----------------|--------------------|
| Hard Disk      | 1.2 TB             |
| RAM Memory     | 8 GB               |
| Processor      | AMD FX-8350        |
| GPU            | GeForce GTX 960    |
| GPU Memory     | 2 GB               |

Table 5.2: ADM laboratory workstation hardware description.

The last important resource is the Chameleon Cloud, which is a configurable experimental online environment for large-scale cloud research. Chameleon has different hardware options available for many different uses. Specifically for THS, the nodes used were called “GPU-100”, and had the software tools presented in Section 5.2. The machines ran the Ubuntu 16.04 LTS OS on bare metal. We trained several models independently of different nodes with the same configuration. For this purpose, we created a custom image in the Chameleon image repository. The hardware specifications of those nodes are:

| <b>Element</b> | <b>Description</b>                               |
|----------------|--|
| Hard Disk      | 207 GB   |
| RAM Memory     | 128 GB   |
| Processor X 2  | Intel(R) Xeon(R)<br>CPU E5-2670 v3 @<br>2.30 GHz |
| GPU            | Tesla P100                                       |
| GPU Memory     | 32 GB  |

Table 5.3: Chameleon cloud nodes hardware description.

## 5.2 Software

We implemented THS using open source software: Hadoop, Yarn, Spark, Hive, Kafka, TensorFlow, Keras, and Scikit-learn. A brief explanation of each one was in Section 4.1. Table 5.4 depicts the specific version of the components.

| <b>Software Package</b> | <b>Version</b> |
|-------------------------|----------------|
| Hadoop & Yarn           | 2.7            |
| Spark                   | 2.1            |
| Hive                    | 2.2            |
| Kafka                   | 0.10.1         |
| TensorFlow              | 1.10           |
| Keras                   | 2.1.6          |
| Scikit-learn            | 0.19.2         |

Table 5.4: Version for software packages used in THS.

## 5.3 Experimental Results

We collected a total of 56,013 tweets from the Twitter streaming API between March 7th and 28th, 2018. The tweets contain at least one of the following medical keywords:

- Zika
- Flu
- Ebola
- Measles
- Diarrhea

We then extracted a random sample of 12,500 tweets for labelling purposes. The labeling process was done in twenty nine days (29) by four members of our team. As mentioned before, we used three class labels: a) 0 - tweet is not about diseases, b) 1 - tweet is related with diseases, and c) 2 - tweet is ambiguous. Table 5.5 shows a distribution of the label classes in our labelled data set. As we can see, the data is unbalanced. In Section 5.3.1 we describe how we handled this situation.

| <b>Class Label</b> | <b>Tweet Count</b> |
|--------------------|--------------------|
| 0                  | 3,850              |
| 1                  | 7,917              |
| 2                  | 733                |

Table 5.5: Distribution of tweets per class label.

### 5.3.1 Experimental Methods

To train our ML models, our training program read the entire data set into memory, randomly shuffled all tweets, and then randomly assigned each tweet into one of three sub-sets:

- **Training set** (60% of the data) - this data set was used to train each ML model.
- **Development set** (20% of the data) - this second data set was used to fit the hyperparameters in the model, and determine which were the best performing candidates.
- **Test set** (20% of the data) - this third data set was used to give an unbiased evaluation of the candidate models and pick the best performing one for the metric at hand.

We used the shuffle functionality in Keras to shuffle training and development data. We also used the scikit-learn built-in support for K-fold cross-validation, but found very little difference between the two approaches.

Since most ML models assume a uniform distribution of examples among the classes present in the data, we had to find a way to adjust our models for the fact that we were working with imbalanced classes. We decided on two approaches to handle this situation. First, we ditched accuracy as our performance metric and instead use precision, recall, and F1 score, explained before in Section 4.4.3. Notice that with an imbalanced class, a classifier might simply always predict in favor of the majority class. Hence, accuracy might not be the most adequate metric. For the validation and test phases, we compute a confusion matrix to collect the performance metrics on each model.

| Class | Penalty Weight |
|-------|----------------|
| 0     | 1.08           |
| 1     | 0.53           |
| 2     | 5.68           |

Table 5.6: Class weight penalties.

The second decision was to use a penalized model approach. Under this scheme, an additional penalty is added to the cost function whenever a model misclassifies, during training, an example that belongs to one of the minority classes. We used the built-in functionality in scikit-learn to estimate class weights penalties for imbalanced datasets. Table 5.6 shows the class weights used for our experiments. As we can see from the figure, the penalty for misclassifying an example in the minority class is substantially larger than that for the majority class. We considered using other approaches for class imbalance, such as oversampling the minority classes, or creating synthetic examples. However, we felt that weight penalization provided the most natural and straightforward method.

### 5.3.2 Finding the Best Models

A total of 1,120 and 1,080 combinations of different hyperparameters were evaluated for RNN and CNN respectively. Each model was trained with the training set composed by 60% of the total data. The algorithm saves a vector array with 8 elements: five slots for the five best, and three slots for the worst three. The validation set, composed by 20% of total data, was used to obtain the metrics explained in Section 4.4.3. For each combination of hyperparameter, the model was trained with the training set, and evaluated with the validation data set. The results obtained from this step were compared with the values in the array, and the best and worst models were updated. When the algorithm ends a text file is generated with the summary of the 8 models. Also, for each model  $M$ , we generated *.h5*, *.json* and, *.txt* files. The *h5* file contains the model architecture, weights, the training configuration (e.g. loss, optimizer), the state of the optimizer. The *json* file contains the model configuration and specification. The *txt* file contains the execution time, the combination of hyperparameters evaluated, the confusion matrix, and the details metrics for each class label  $\{0,1,2\}$ .

To run the RNN algorithm we used a model composed by two LSTM layers of 50 hidden units and an attention layer between them. On the other hand, to run the CNN algorithm we used a model composed by an inception layer that contains a 4 convolutional layers, as explained in 4.7. More details of the evaluated architecture are presented in the next sections. Once the top 5 best combinations are obtained for each architecture, other models composed with different elements were trained and evaluated with the *test set* (holdout set). These different elements were explained in Tables 4.4 and 4.5 respectively. Hence, each different model architecture evaluated has the files and metrics calculated for the 5 best combinations. To find the final best combination for RNN and CNN, there a comparison of the performance metrics in the test set was made.

### 5.3.3 Final Results on RNN

All the RNN models tested were run with this set of hyperparameters: a learning rate of 0.0001, 5 epochs of training, a batch size of 32, RMSprop was the optimizer, 0.3 rate for the first dropout layer, 0.1 rate for the second dropout layer, and one 32-units dense layer after the second LSTM/GRU layer, and before the softmax output layer. These hyperparameters were found with the procedure described in the previous section.

Table 5.7 shows the results for the precision metrics for class 1 - tweets related with a disease. For the sake of clarity, we only present this class. In addition, it is the most

relevant class for the purpose of using Twitter Streaming API to detect conversations about diseases. Notice that the best performing model, *2 LSTM Attention 100 Units* has a precision of 96%.

| Abbreviation                  | Precision |
|-------------------------------|-----------|
| 2 LSTM Attention 50 Units     | 0.87      |
| 2 LSTM No Attention 50 Units  | 0.83      |
| 2 LSTM Attention 100 Units    | 0.96      |
| 2 LSTM No Attention 100 Units | 0.88      |
| 2 GRU Attention               | 0.64      |
| 2 GRU No Attention            | 0.63      |
| LSTM GRU Attention            | 0.64      |
| LSTM GRU No Attention         | 0.83      |
| GRU LSTM Attention            | 0.60      |
| GRU LSTM No Attention         | 0.60      |

Table 5.7: Precision results per RNN model.

Table 5.8 shows the results for the recall metrics for class 1. Notice that the best performing model, *LSTM GRU No Attention* has a recall of 89%. Also, notice that various entries have recall of 0. This means that the model did not converge.

| Abbreviation                  | Recall |
|-------------------------------|--------|
| 2 LSTM Attention 50 Units     | 0.78   |
| 2 LSTM No Attention 50 Units  | 0.79   |
| 2 LSTM Attention 100 Units    | 0.34   |
| 2 LSTM No Attention 100 Units | 0.69   |
| 2 GRU Attention               | 0.54   |
| 2 GRU No Attention            | 0.26   |
| LSTM GRU Attention            | 0.0    |
| LSTM GRU No Attention         | 0.89   |
| GRU LSTM Attention            | 0.0    |
| GRU LSTM No Attention         | 0.0    |

Table 5.8: Recall results per RNN model.

Table 5.9 shows the results for the F1 Score metrics for class 1. Notice that the best

performing model, *LSTM GRU No Attention*, has a F1 Score of 86%.

| Abbreviation                  | F1 Score |
|-------------------------------|----------|
| 2 LSTM Attention 50 Units     | 0.82     |
| 2 LSTM No Attention 50 Units  | 0.81     |
| 2 LSTM Attention 100 Units    | 0.50     |
| 2 LSTM No Attention 100 Units | 0.78     |
| 2 GRU Attention               | 0.59     |
| 2 GRU No Attention            | 0.37     |
| LSTM GRU Attention            | 0.78     |
| LSTM GRU No Attention         | 0.86     |
| GRU LSTM Attention            | 0.75     |
| GRU LSTM No Attention         | 0.75     |

Table 5.9: F1 Score results per RNN model.

Table 5.10 shows the running time for training and validation for each RNN model. Notice that the fastest model, *2 GRU Attention*, ran in 4.44 minutes on the node with GPU located in the Chameleon Cloud.

| Abbreviation                  | Execution time (mins) |
|-------------------------------|-----------------------|
| 2 LSTM Attention 50 Units     | 4.62                  |
| 2 LSTM No Attention 50 Units  | 4.77                  |
| 2 LSTM Attention 100 Units    | 4.70                  |
| 2 LSTM No Attention 100 Units | 4.86                  |
| 2 GRU Attention               | 4.44                  |
| 2 GRU No Attention            | 5.17                  |
| LSTM GRU Attention            | 4.48                  |
| LSTM GRU No Attention         | 4.45                  |
| GRU LSTM Attention            | 5.49                  |
| GRU LSTM No Attention         | 5.81                  |

Table 5.10: Execution time per RNN model.

### 5.3.4 Final Results on CNN

All the RNN models tested were run with this set of hyperparameters: a learning rate of 0.0001, 40 epochs of training, a batch size of 32, RMSprop was the optimizer, 64 filters were used in each convolution, 0.1 rate for the dropout layer, and a 128-units dense layer after the inception layer(s) and before the softmax output layer.

Table 5.11 shows the results for the precision metrics for class 1. For the sake of clarity, we only present this class. In addition, it is the most relevant class for the purpose of using Twitter Streaming to detect conversations about diseases. Notice that the best performing model, *4 inception 3x3*, has a precision of 79%.

| Abbreviation    | Precision |
|-----------------|-----------|
| 1 inception 3x3 | 0.77      |
| 1 inception 5x5 | 0.78      |
| 4 inception 3x3 | 0.79      |
| 4 inception 5x5 | 0.76      |

Table 5.11: Precision results per CNN model.

Table 5.12 shows the results for the recall metrics for class 1. Notice that the best performing model, *1 inception 3x3*, has a recall of 91%.

| Abbreviation    | Recall |
|-----------------|--------|
| 1 inception 3x3 | 0.91   |
| 1 inception 5x5 | 0.85   |
| 4 inception 3x3 | 0.82   |
| 4 inception 5x5 | 0.90   |

Table 5.12: Recall results per CNN model.

Table 5.13 shows the results for the F1 Score metrics for class 1. Notice that the best performing model, *1 inception 3x3* has a F1 Score of 83%.

| Abbreviation    | F1 Score |
|-----------------|----------|
| 1 inception 3x3 | 0.83     |
| 1 inception 5x5 | 0.82     |
| 4 inception 3x3 | 0.81     |
| 4 inception 5x5 | 0.82     |

Table 5.13: F1 Score results per CNN model.

Table 5.14 shows the running time for training and validation for each CNN model. Notice that the fastest model, *1 inception 3x3*, runs in 4.65 minutes.

| Abbreviation    | Execution time (mins) |
|-----------------|-----------------------|
| 1 inception 3x3 | 4.65                  |
| 1 inception 5x5 | 5.33                  |
| 4 inception 3x3 | 31.19                 |
| 4 inception 5x5 | 37.44                 |

Table 5.14: Execution time per CNN model.

### 5.3.5 Discussion of Results

In the results of the RNN model we concluded that the best performing model has a precision of 96%, this models contains two LSTM layers with an attention layer between them. Each LSTM layer contains one hundred of hidden units to process and modify the weights. Notice that the worst precision rate was 60% with a combination of GRU layer followed by LSTM layer tested with attention layer and without it. For the recall metric the best rate reached was 89%, the model which give this metric has a LSTM layer follow by GRU layer without attention layer. The worst model for this metric is conformed by two GRU layers without attention layer too and gives a 26%. The best model for the F1 Score metric consist of a LSTM followed by a GRU layer with a 86%. The worst model for the F1 score metric has a value of 37%, and consists of two GRU layers without an attention layer. The fastest model was trained and evaluated with the metrics in 4.45 minutes , this was a LSTM layer followed by GRU layer without attention. The slowest model took 5.81 minutes, and was a GRU layer followed by LSTM without attention layer.

All the CNN models' metrics are close among them, and not very distant like the RNN models. The best precision obtained was 79%, this model is shaped by four inception layers with a batch size of three by three. The model which provides the worst precision

has four inception layers with batch size of five by five, giving a precision of 76%. In the recall metric the best model gives a 91%, this model contains one inception layer with batch size of three by three. Notice that the worst recall rate was 82% with a model formed by four inception layers with batch size of three by three. The best F1 score is given by a model with one inception layer with batch size of three by three, performing a rate of 83%. The worst model has four inception layers with batch size of three by three and giving 80%. The fastest model was trained and evaluated with the metrics in 4.65 minutes, the model contained one inception layer and a batch size of three by three. The slowest model took 37.43 minutes, this model has 4 inception layers with batch size of five by five.

In the comparison between RNN and CNN model, we can see that the RNN model metrics are better in Precision, F1 Score, and execution time versus the CNN models. Nonetheless, the CNN model is better just in the Recall metric. Another important factor to note is the variation between the results in each table. For example the Recall Table 5.8 for RNN varied between 26% and 89%. Instead, the CNN models Table 5.12 varied between 82% and 91%. Thus CNN tend to provide less variable models versus RNN architectures. The execution time for the CNN model is longer than RNN, it was almost 6 times more than the worst RNN execution time model. The worst time for CNN model was expect by due of the number of epochs, inception layers, and the size of the kernel size.

## Chapter 6

# Conclusion and Future Work

In this work, we have presented the Twitter Health Surveillance (THS) application framework. THS is designed as an integrated platform to help health officials collect tweets, determine if they are related with a medical condition, extract metadata out of them, and create a warehouse that can be used to further analyze the data. THS is built atop open source tools and provides the following value added services: Data Acquisition, Tweet Classification, and Big Data Warehousing. We presented the infrastructure necessary to feed the tweets into a machine learning pipeline that can efficiently learn to classify the tweets as being related or not with diseases, and then use those models in a production environment. This infrastructure was built atop the Hadoop big data tool set, Google's Tensorflow and Keras. All of this information can be provided to public health officials through a web-based dashboard.

We used neural networks, both recurrent and convolutional, for the classification process. The input to these the ML algorithms is a training set that contains tweets with various medical terms and a label for each tweet. The label indicates if the tweet is related or not to a medical condition.

In order to validate THS, we have created a collection of 12,500 labelled tweets. These tweets contain one or more target medical terms, and the labels indicate if the tweet is related or not to a medical condition. Specifically, each tweet is labelled into one of three classes: a) class 0 - does not talk about medical condition, b) class 1 - talks about a medical condition, and c) class 2 - ambiguous. We used this collection to test various models based on LSTM and GRU for RNN, and on inception modules for CNN. Our experiments show that we can classify tweets with 96% precision, 91% recall, and 86% F1 score. These results compare favorably with recent research on this area, and show the promise of our THS system as a tool to help detect real disease chat on Twitter. Our

experiments show that the RNN model performed better with Precision, F1 Score, and execution time versus the CNN models. Nonetheless, the CNN model is better with the Recall metric. We also found that CNN tend to provide less variable models, in terms of Precision and Recall, versus RNN architectures. The execution time for the CNN model is longer than RNN, it was almost 6 times more than the worst RNN execution time model. The worst time for CNN model was expect due of the number of epochs, inception layers, and the kernel size.

Future work will be focused on training the machine learning models with more data which enable us to obtain new results, and compare with the previous models. Also, we will explore how to continuously update the models as new data is captured. In addition, we shall design larger architectures for RNN and CNN to determine if these improve the performance of the system. Furthermore, models with unsupervised or semi-supervised algorithms of machine learning could be tested. Example algorithms are k-means, DBSCAN, anomaly detection, among others.

# References

- [1] Jeff Schultz. *How Much Data is Created on the Internet Each Day*. Oct. 2017. URL: <https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/>.
- [2] Elizabeth D Liddy. “Natural Language Processing”. In: *Encyclopedia of Library and Information Science 2* (2005). DOI: 978-0-8493-3894-6. URL: <https://surface.syr.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com.co/&httpsredir=1&article=1019&context=cnlp>.
- [3] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.
- [4] Tom M Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997, p. 432. ISBN: 0070428077. URL: <https://www.cs.ubbcluj.ro/%7B~%7Dgabis/ml/ml-books/McGrawHill%20-%20Machine%20Learning%20-Tom%20Mitchell.pdf>.
- [5] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*, New York, NY, USA: Dover Publications, Inc., 1974. ISBN: 0486612724.
- [6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *AISTATS 15* (2011), pp. 315–323. URL: <http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>.
- [7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 115–116. ISBN: 0387310738. URL: <https://dl.acm.org/citation.cfm?id=1162264>.
- [8] Rob J Hyndman and George Athanaffdfdfdsopuffdfdfdos. *Neural network models*. OTexts, 2013. Chap. 9.3 Neural, p. 292. ISBN: 0987507109. URL: <https://www.otexts.org/fpp/9/3>.

- [9] Jfffdffdrngen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Elsevier* (Oct. 2014), 85fffdfffdffd117. URL: <https://www.sciencedirect.com/science/article/pii/S0893608014002135>.
- [10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [11] Andrea Vedaldi Max Jaderberg Karen Simonyan and Andrew Zisserman. “Deep structured output learning for unconstrained text recognition”. In: *International Conference on Learning Representations* (2015). URL: <https://arxiv.org/pdf/1412.5903.pdf>.
- [12] Bo Wu Yuval Netzer Tao Wang Adam Coates Alessandro Bissacco and Andrew Y. Ng. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: (Jan. 2011). URL: [https://www-cs.stanford.edu/~twangcat/papers/nips2011\\_housenumbers.pdf](https://www-cs.stanford.edu/~twangcat/papers/nips2011_housenumbers.pdf).
- [13] Sacha Arnoud Ian J. Goodfellow Yaroslav Bulatov Julian Ibarz and Vinay Shet. “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks”. In: (Apr. 2014). URL: <https://arxiv.org/pdf/1312.6082.pdf>.
- [14] Max Jaderberg Karen Simonyan Andrea Vedaldi Andrew Zisserman. “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition”. In: (Dec. 2014). URL: <https://arxiv.org/pdf/1406.2227.pdf>.
- [15] Awni Hannun Carl Case Jared Casper Bryan Catanzaro Greg Diamos Erich Elsen Ryan Prenger Sanjeev Satheesh Shubho Sengupta Adam Coates Andrew Y. Ng. “Deep Speech: Scaling up end-to-end speech recognition”. In: (Dec. 2014). URL: <https://arxiv.org/pdf/1412.5567.pdf>.
- [16] Jiquan Ngiam Aditya Khosla Mingyu Kim Juhan Nam Honglak Lee Andrew Y. Ng. In: *ICML’11 Proceedings of the 28th International Conference on International Conference on Machine Learning* (June 2011), pp. 689–696. URL: <http://ai.stanford.edu/~ang/papers/icml11-MultimodalDeepLearning.pdf>.
- [17] Tetsuya Nasukawa and Jeonghee Yi. “Sentiment analysis”. In: *Proceedings of the international conference on Knowledge capture - K-CAP ’03* March (2003), p. 70. DOI: 10.1145/945645.945658. URL: <http://portal.acm.org/citation.cfm?doid=945645.945658>.

- [18] Lei Zhang and Bing Liu. “Sentiment Analysis and Opinion Mining”. In: *Encyclopedia of Machine Learning and Data Mining* May (2016), pp. 1–10. ISSN: 1947-4040. DOI: 10.1007/978-1-4899-7502-7\_907-1. arXiv: 1003.5699. URL: [http://link.springer.com/10.1007/978-1-4899-7502-7%7B%5C\\_%7D907-1](http://link.springer.com/10.1007/978-1-4899-7502-7%7B%5C_%7D907-1).
- [19] Wang Hao et al. “A System for Real-time Twitter Sentiment Analysis of 2012 U . S . Presidential Election Cycle”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics* July (2012), pp. 115–120.
- [20] Bjarke Felbo et al. “Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm”. In: (2016). arXiv: arXiv:1708.00524v2.
- [21] Jason Weston, N E C Labs America, and Independence Way. “A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning”. In: (2008).
- [22] Gaël Guibon and Patrice Bellot. “From Emojis to Sentiment Analysis”. In: (2016).
- [23] Ben Eisner et al. “emoji2vec : Learning Emoji Representations from their Description”. In: (2016). arXiv: arXiv:1609.08359v2.
- [24] J. Parker Y. Wei A. Yates O. Frieder and N. Goharian. “A framework for detecting public health trends with twitter”. In: Proc. of 2013 IEEE/ACM ASONAM Conf. (2013), pp. 556–563.
- [25] Mizuki MORITA Aramaki Eiji Sachiko MASKAWA. “Twitter Catches The Flu Detecting Influenza Epidemics using Twitter”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (2011), pp. 1568–1576. URL: <http://www.aclweb.org/anthology/D11-1145>.
- [26] Aron Culotta. “Detecting influenza outbreaks by analyzing Twitter messages”. In: *Proceeding SOMA '10 Proceedings of the First Workshop on Social Media Analytics* (2010), pp. 115–122. ISSN: 03050270. DOI: 10.1145/1964858.1964874. arXiv: 1007.4748. URL: <http://arxiv.org/abs/1007.4748>.
- [27] Ernesto Diaz-Aviles et al. “Towards personalized learning to rank for epidemic intelligence based on social media streams”. In: *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion* April (2012), pp. 495–496. ISSN: 0006-3223. DOI: 10.1145/2187980.2188094. arXiv: arXiv:1203.1378v1. URL: <http://dl.acm.org/citation.cfm?doid=2187980.2188094>.

- [28] Michael J. Paul and Mark Dredze. “Discovering health topics in social media using topic models”. In: *PLoS ONE* 9.8 (2014). ISSN: 19326203. DOI: 10.1371/journal.pone.0103408.
- [29] Amira Ghenai and Yelena Mejova. “Catching Zika Fever: Application of Crowdsourcing and Machine Learning for Tracking Health Misinformation on Twitter”. In: *CoRR* abs/1707.03778 (2017). arXiv: 1707.03778. URL: <http://arxiv.org/abs/1707.03778>.
- [30] Xiang Ji, Soon Ae Chun, and James Geller. “Monitoring public health concerns using twitter sentiment classifications”. In: *Proceedings - 2013 IEEE International Conference on Healthcare Informatics, ICHI 2013*. 2013, pp. 335–344. ISBN: 9780769550893. DOI: 10.1109/ICHI.2013.47.
- [31] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. “A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data”. In: *SIGKDD Explor. Newsl.* 6.1 (June 2004), pp. 20–29. ISSN: 1931-0145. DOI: 10.1145/1007730.1007735. URL: <http://doi.acm.org/10.1145/1007730.1007735>.
- [32] H. He and E. A. Garcia. “Learning from Imbalanced Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284. ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.239.
- [33] The Apache Software Foundation. *Introduction*. 2018. URL: <http://kafka.apache.org/intro.html>.
- [34] The Apache Software Foundation. *Spark Overview*. 2018. URL: <https://spark.apache.org/docs/latest/>.
- [35] Matei Zaharia et al. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”. In: *Nsdi* (2012), pp. 2–2. ISSN: 00221112. DOI: 10.1111/j.1095-8649.2005.00662.x. arXiv: EECS-2011-82. URL: <https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf>.
- [36] The Apache Software Foundation. *Apache Software Foundation*. 2018. URL: <https://cwiki.apache.org/confluence/display/Hive/Home>.
- [37] The Apache Software Foundation. *HDFS Architecture Guide*. 2008. URL: [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).

- [38] Arshdeep Bahga and V. Madiseti. *Big data science analytics : a hands-on approach*. 2016. ISBN: 9780996025546. URL: [http://www.worldcat.org/title/big-data-science-analytics-a-hands-on-approach/oclc/953867348&referer=brief\\_results](http://www.worldcat.org/title/big-data-science-analytics-a-hands-on-approach/oclc/953867348&referer=brief_results).
- [39] The Apache Software Foundation. *Apache Hadoop YARN*. 2018. URL: <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>.
- [40] Google. *Get Started with TensorFlow*. 2015. URL: <https://www.tensorflow.org/tutorials/>.
- [41] Keras. *Keras: The Python Deep Learning library*. 2018. URL: <https://keras.io/>.
- [42] *The Apache Software Foundation*. 2018. URL: <http://www.apache.org/>.
- [43] Fred Morstatter et al. “Is the Sample Good Enough? Comparing Data from Twitter’s Streaming API with Twitter’s Firehose”. In: (2013). ISSN: 16113349. DOI: 10.1007/978-3-319-05579-4\_10. URL: <http://arxiv.org/abs/1306.5204>.
- [44] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: <http://www.aclweb.org/anthology/D14-1162>.
- [45] Yoav Goldberg and Omer Levy. “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method”. In: 2 (2014), pp. 1–5. ISSN: 0003-6951. DOI: 10.1162/jmlr.2003.3.4-5.951. arXiv: 1402.3722. URL: <http://arxiv.org/abs/1402.3722>.
- [46] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: (2013), pp. 1–12. ISSN: 15324435. DOI: 10.1162/153244303322533223. arXiv: 1301.3781. URL: <http://arxiv.org/abs/1301.3781>.
- [47] Mikolov Tomas et al. “Efficient Estimation of Word Representations in Vector Space”. In: *CoRR* abs/1301.3781 (2013). arXiv: 1301.3781. URL: <http://arxiv.org/abs/1301.3781>.
- [48] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2014). arXiv: 1409.0473. URL: <http://arxiv.org/abs/1409.0473>.

- [49] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 07-12-June-2015 (2015)*, pp. 1–9. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.
- [50] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: (2015). ISSN: 08866236. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567. URL: <http://arxiv.org/abs/1512.00567>.

# Appendices

# Appendix A

## Experimental Results for Classes 0 and 2

### A.1 RNN Results for Class 0

Table A.1 shows the results for the precision metrics for class 0 - tweets non related with a disease.

| Abbreviation                  | Precision |
|-------------------------------|-----------|
| 2 LSTM Attention 50 Units     | 0.95      |
| 2 LSTM No Attention 50 Units  | 0.90      |
| 2 LSTM Attention 100 Units    | 0.91      |
| 2 LSTM No Attention 100 Units | 0.94      |
| 2 GRU Attention               | 0.00      |
| 2 GRU No Attention            | 0.39      |
| LSTM GRU Attention            | 0.00      |
| LSTM GRU No Attention         | 0.00      |
| GRU LSTM Attention            | 0.00      |
| GRU LSTM No Attention         | 0.00      |

Table A.1: Precision results per RNN model for class 0.

Table A.2 shows the results for the recall metrics for class 0 - tweets non related with a disease.

| <b>Abbreviation</b>           | <b>Recall</b> |
|-------------------------------|---------------|
| 2 LSTM Attention 50 Units     | 0.24          |
| 2 LSTM No Attention 50 Units  | 0.40          |
| 2 LSTM Attention 100 Units    | 0.40          |
| 2 LSTM No Attention 100 Units | 0.28          |
| 2 GRU Attention               | 0.00          |
| 2 GRU No Attention            | 0.03          |
| LSTM GRU Attention            | 0.00          |
| LSTM GRU No Attention         | 0.00          |
| GRU LSTM Attention            | 0.00          |
| GRU LSTM No Attention         | 0.00          |

Table A.2: Recall results per RNN model for class 0.

Table A.3 shows the results for the recall metrics for class 0 - tweets non related with a disease.

| <b>Abbreviation</b>           | <b>F1 Score</b> |
|-------------------------------|-----------------|
| 2 LSTM Attention 50 Units     | 0.39            |
| 2 LSTM No Attention 50 Units  | 0.56            |
| 2 LSTM Attention 100 Units    | 0.55            |
| 2 LSTM No Attention 100 Units | 0.44            |
| 2 GRU Attention               | 0.00            |
| 2 GRU No Attention            | 0.05            |
| LSTM GRU Attention            | 0.00            |
| LSTM GRU No Attention         | 0.00            |
| GRU LSTM Attention            | 0.00            |
| GRU LSTM No Attention         | 0.00            |

Table A.3: F1 Score results per RNN model for class 0.

## A.2 RNN Results for Class 2

Table A.4 shows the results for the precision metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b>           | <b>Precision</b> |
|-------------------------------|------------------|
| 2 LSTM Attention 50 Units     | 0.05             |
| 2 LSTM No Attention 50 Units  | 0.04             |
| 2 LSTM Attention 100 Units    | 0.04             |
| 2 LSTM No Attention 100 Units | 0.04             |
| 2 GRU Attention               | 0.03             |
| 2 GRU No Attention            | 0.06             |
| LSTM GRU Attention            | 0.00             |
| LSTM GRU No Attention         | 0.03             |
| GRU LSTM Attention            | 0.00             |
| GRU LSTM No Attention         | 0.00             |

Table A.4: Precision results per RNN model for class 2.

Table A.5 shows the results for the recall metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b>           | <b>Recall</b> |
|-------------------------------|---------------|
| 2 LSTM Attention 50 Units     | 0.52          |
| 2 LSTM No Attention 50 Units  | 0.30          |
| 2 LSTM Attention 100 Units    | 0.88          |
| 2 LSTM No Attention 100 Units | 0.49          |
| 2 GRU Attention               | 0.47          |
| 2 GRU No Attention            | 0.83          |
| LSTM GRU Attention            | 0.00          |
| LSTM GRU No Attention         | 0.47          |
| GRU LSTM Attention            | 0.00          |
| GRU LSTM No Attention         | 0.00          |

Table A.5: Recall results per RNN model for class 2.

Table A.6 shows the results for the recall metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b>           | <b>F1 Score</b> |
|-------------------------------|-----------------|
| 2 LSTM Attention 50 Units     | 0.09            |
| 2 LSTM No Attention 50 Units  | 0.07            |
| 2 LSTM Attention 100 Units    | 0.08            |
| 2 LSTM No Attention 100 Units | 0.07            |
| 2 GRU Attention               | 0.06            |
| 2 GRU No Attention            | 0.11            |
| LSTM GRU Attention            | 0.00            |
| LSTM GRU No Attention         | 0.06            |
| GRU LSTM Attention            | 0.00            |
| GRU LSTM No Attention         | 0.00            |

Table A.6: F1 Score results per RNN model for class 2.

### A.3 CNN Results for Class 0

Table A.7 shows the results for the precision metrics for class 0 - tweets non related with a disease.

| <b>Abbreviation</b> | <b>Precision</b> |
|---------------------|------------------|
| 1 inception 3x3     | 0.76             |
| 1 inception 5x5     | 0.71             |
| 4 inception 3x3     | 0.67             |
| 4 inception 5x5     | 0.71             |

Table A.7: Precision results per CNN model for class 0.

Table A.8 shows the results for the recall metrics for class 0 - tweets non related with a disease.

| <b>Abbreviation</b> | <b>Recall</b> |
|---------------------|---------------|
| 1 inception 3x3     | 0.47          |
| 1 inception 5x5     | 0.55          |
| 4 inception 3x3     | 0.55          |
| 4 inception 5x5     | 0.51          |

Table A.8: Recall results per CNN model for class 0.

Table A.9 shows the results for the F1 Score metrics for class 0 - tweets non related with a disease.

| <b>Abbreviation</b> | <b>F1 Score</b> |
|---------------------|-----------------|
| 1 inception 3x3     | 0.58            |
| 1 inception 5x5     | 0.62            |
| 4 inception 3x3     | 0.60            |
| 4 inception 5x5     | 0.59            |

Table A.9: F1 Score results per CNN model for class 0.

## A.4 CNN Results for Class 2

Table A.10 shows the results for the precision metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b> | <b>Precision</b> |
|---------------------|------------------|
| 1 inception 3x3     | 0.07             |
| 1 inception 5x5     | 0.08             |
| 4 inception 3x3     | 0.04             |
| 4 inception 5x5     | 0.07             |

Table A.10: Precision results per CNN model for class 2.

Table A.11 shows the results for the recall metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b> | <b>Recall</b> |
|---------------------|---------------|
| 1 inception 3x3     | 0.10          |
| 1 inception 5x5     | 0.13          |
| 4 inception 3x3     | 0.09          |
| 4 inception 5x5     | 0.03          |

Table A.11: Recall results per CNN model for class 2.

Table A.12 shows the results for the F1 Score metrics for class 2 - tweets which are ambiguous.

| <b>Abbreviation</b> | <b>F1 Score</b> |
|---------------------|-----------------|
| 1 inception 3x3     | 0.09            |
| 1 inception 5x5     | 0.10            |
| 4 inception 3x3     | 0.06            |
| 4 inception 5x5     | 0.04            |

Table A.12: F1 Score results per CNN model for class 2.

# Appendix B

## GitHub Repositories

The GitHub repositories of the big data and machine learning daemon are available upon request at [cristian.garzon@upr.edu](mailto:cristian.garzon@upr.edu). The following sections contain the links.

### B.1 Big Data Platform

<https://github.com/THSUPRM/bigdata/tree/master/python>

#### B.1.1 Machine Learning Platform

<https://github.com/THSUPRM/bigdata/tree/master/DetectDiseaseTHS/th>