

Finding Similar Tweets Within Health Related Topics

By

Danny Gilberto Villanueva Vega

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

2019

Approved by:

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

Date

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

Date

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

Date

Ivette Cruzado, Ph.D.
Graduate School Representative

Date

Bienvenido J Velez Rivera, Ph.D.
Dean of Engineering

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

Finding Similar Tweets Within Health Related Topics

Social networks have become a very important means to facilitate the creation and sharing of information, ideas, news, and opinions on many topics. They also provide real-time information on sales, marketing, politics, natural disasters, and crisis situations, among others. These networks include Facebook, Twitter, WhatsApp, and Instagram, to name a few. In this work, we shall focus our efforts on the Twitter social network. This network provides a mechanism for people to express their views using short messages (i.e., 280 characters) called *tweets*. In this project, we investigate and implement text similarity neural network models in such a way that we can: 1) know if they are related or not with a disease, 2) group similar tweets to those that we have already captured, analyzed or stored, and 3) find similarity index between tweets using different learning algorithms. We based our work on, semantic similarity approaches and text similarity measures using Deep Learning algorithms to deliver reliable information about health-related topics.

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 Ciencias en Ingeniería de Computadoras

Encontrando tweets similares en temas relacionados con la salud

Las redes sociales se han convertido en un medio muy importante para crear y compartir información, ideas, noticias y opiniones sobre muchos temas. Estas también proporcionan información en tiempo real sobre ventas, mercadotecnia, política, desastres naturales y situaciones de crisis, entre otros. Estas redes incluyen Facebook, Twitter, WhatsApp e Instagram, por nombrar algunas. En este trabajo, centraremos nuestros esfuerzos en la red social Twitter. Esta red proporciona un mecanismo para que las personas expresen sus puntos de vista mediante mensajes cortos (no más de 280 caracteres) llamados *tweets*. En este proyecto, investigamos e implementamos modelos de redes neuronales de similitud de texto de manera que podamos: 1) saber si están relacionados o no con una enfermedad, 2) agrupar tweets similares a los que ya hemos capturado, analizado o almacenado y 3) encontrar el índice de similitud entre los tweets que utilizan diferentes algoritmos de aprendizaje. Basamos nuestro trabajo en los enfoques de similitud semántica y las medidas de similitud de texto utilizando algoritmos de “Deep Learning” para proporcionar información confiable sobre temas relacionados a la salud.

Copyright © 2019

by

Danny Gilberto Villanueva Vega

DEDICATION

To my Mom, Carin Vega Pérez. To my sister, Emyli S Rodriguez Vega.

Acknowledgments

First of all thank God for giving me health, strength and the desire to persevere forever. I also want to thank my mother Carin and my sister Emily for the love, affection and unconditional support they have always given me, without which it would have been more difficult to achieve this goal.

I would like to express my sincere gratitude to my adviser Manuel Rodriguez, for the opportunity to work with him, for his patience, motivation, teaching during the whole process of the research and his support during my master's studies. His guidance and advice was very helpful in all aspects of the research and thesis presentation.

I would also like to thank the 54 labelers, which were students from University of Puerto Rico Mayagüez Campus (UPRM) that were enrolled in the course "Introduction to Database Systems"

This research is supported by the United States (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 Cloud supported by the National Science Foundation (NSF).

Contents

Abstract	ii
Abstract (Spanish)	iii
Acknowledgment	vi
List of Figures	ix
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Key Innovation	3
1.4 Contributions	4
1.4.1 Technical Contributions	4
1.4.2 Contributions to Society	6
1.5 Outline	6
2 Literature Review	7
2.1 Introduction	7
2.2 Machine Learning	8

2.2.1	Supervised Learning	8
2.2.2	Unsupervised Learning	10
2.3	Neural Networks	11
2.3.1	Sigmoid function	13
2.3.2	Hyperbolic tangent function	13
2.3.3	Rectified linear unit function	13
2.3.4	Softmax function	14
2.3.5	Connecting Layers	14
2.3.6	Parameters and Hyperparameters	15
2.3.7	Data Splitting	15
2.4	Deep Learning with Neural Networks	16
2.4.1	Fully Connected Networks	16
2.4.2	Convolutional Neural Networks	16
2.4.3	Recurrent Neural Networks	17
2.5	Natural Language Processing	18
2.6	Text Similarity	21
2.6.1	Lexical Similarity	21
2.6.2	Semantic Similarity	22
3	Problem Formulation	24
3.1	Description	24
3.1.1	Examples	25
3.2	Formalization	26
3.2.1	Scope of the Model	27
3.2.2	Use in Production	27
3.3	Data Processing Architecture	28

4	System Architecture and Organization	29
4.1	THS System Overview	29
4.2	Strategy to Obtain the Model for Similarity	31
4.3	Labeling	32
4.4	Building the Dataset for Training	32
4.5	Architecture of the Learning Model	34
4.5.1	Data Setup	35
4.5.2	Combiner Layer	36
4.5.3	Relevance Layer	41
4.5.4	Classification Layer	42
4.6	Metrics	42
5	Performance Evaluation	43
5.1	Hardware	43
5.2	Software	44
5.3	Data Collection for Training Models	46
5.4	Data Pre-processing	47
5.5	Data Labeling	48
5.5.1	Disease-Related Labeling	48
5.5.2	Labeling for Level of Relevance (rank)	49
5.6	Experiment Setup	50
5.6.1	Experimental Methods	52
5.6.2	Results for Models using RNN at the Combiner Layer	52
5.6.3	Results for Models using CNN at the Combiner Layer	56
5.6.4	Discussion of Results	58
6	Conclusion and Future Work	60

CONTENTS

CONTENTS

Bibliography	62
Appendices	66
A GitHub Repositories	67
A.1 Big Data Platform	67
A.1.1 Machine Learning Platform	67

List of Abbreviations

AI Artificial Intelligence

API Application Program Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma Separated Values

CUDA Compute Unified Device Architecture

DL Deep Learning

GPU Graphic Processing Unit

GRU Gated Recurrent Unit

HDFS Hadoop Distributed File System

JSON JavaScript Object Notation

LSTM Long Short-Term Memory

ML Machine Learning

NIH National Institutes of Health

NLP Natural Language Processing

NLTK Natural Language Toolkit

NSF National Science Foundation

OS Operating System

CONTENTS

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SQL Structured Query Language

THS Twitter Health Surveillance

UPRM University of Puerto Rico Mayagüez Campus

US United States

YARN Yet Another Resource Negotiator

CONTENTS

List of Figures

2.1	Supervised Learning Workflow.	9
2.2	Unsupervised Learning Workflow.	11
2.3	A Simple Mathematical Neuron Representation as depicted in [1].	12
2.4	Feed Forward Network	14
2.5	Natural Language Process Stages	19
3.1	Data Processing Architecture	28
4.1	Twitter Health Surveillance (THS) Processing Architecture	31
4.2	Model Architecture	35
4.3	Auxiliary Input Representation	36
4.4	Regular LSTM Network	38
4.5	BiLSTM Network	39
4.6	Inception Module as depicted in [36].	40
4.7	Inception Convolutional Neural Network (CNN)	40
4.8	Relevance Layer	41
4.9	Classification Layer	42
5.1	THS Cluster	43
5.2	Data Collection Process	46
5.3	Data Pre-processing	47

5.4 Model Setup Overview 51

List of Tables

4.1	Hyperparameter Combinations	37
5.1	THS Cluster Node	44
5.2	Chameleon Cloud Custom Node	44
5.3	Big Data Tools in THS System	45
5.4	Version of Software in THS Nodes	46
5.5	Triplets Distribution	52
5.6	Accuracy for Regular LSTM	53
5.7	Loss Metrics for Regular LSTM	54
5.8	Accuracy Bidirectional Network	55
5.9	Loss Bidirectional Network	56
5.10	Accuracy CNN Model	57
5.11	Loss CNN Model	58

Chapter 1

Introduction

1.1 Motivation

Social networks have become a very important means to facilitate the creation and sharing of information, ideas, news, and opinions on many topics. They also provide real-time information on sales, marketing, politics, natural disasters, and crisis situations, among others. These networks include Facebook, Twitter, WhatsApp, and Instagram, to name a few.

In this work, we shall focus our efforts on the Twitter social network. This network provides a mechanism for people to express their views using short messages (i.e., 280 characters) called *tweets*. Users of this network can find each other messages without the need of becoming “friends”, as it happens in other networks. The analysis of these tweets can enable us to understand the current situation regarding certain topics, for example, discussions related to medical topics (e.g., “flu”). Users can monitor and find patterns that give information about some type of disease being discussed in the social network. In addition, it is possible to detect the sentiment, “mood” of the people discussing a specific topic.

In the health care domain, searching text in social medias, blogs and newspapers can provide clues about the diseases that are being talked about by citizens of a region. For

example, an increase in messaging related with a disease (e.g., Measles) could indicate that people are concerned due to some outbreak and are looking for more information about symptoms and treatments.

For the analysis of all this available information it is *necessary to group or categorize the text along similarities in structure and/or meaning*. However, this is a challenging task, due to the complexity/ambiguity introduced by spelling errors or the use of informal language (“slang”). In the case of tweets, the small size of the message often makes it difficult to analyze without the context provided by previous messages or user interactions. Making use of the data stored in the THS System from the UPRM, as one of our sources, it is possible to process all the information easily and quickly, and use it to analyze and process the data using Machine Learning (ML) algorithms, a popular branch of Artificial Intelligence (AI) systems. In particular, deep neural network models can be trained to learn to detect whether tweets are related to medical conditions. In addition, these models can be trained to find semantic similarity in a collection of tweets.

In this project, we investigate and implement text similarity neural network models in such a way that we can: 1) know if they are related or not with a disease, 2) group similar tweets to those that we have already captured, analyzed or stored, and 3) find similarity index between tweets using different learning algorithms. We based our work on, semantic similarity approaches and text similarity measures using Deep Learning (DL) algorithms to deliver reliable information to the end user about health-related topics.

1.2 Objectives

Our work is rooted in our THS project at UPRM, whose goal is to monitor tweets related with medical conditions, and detect instances where a disease is actively being discussed in a given region. The main problem that we want to tackle in this project is the development of DL models that can evaluate the similarity between tweets. Specifically, given an

example tweet T_p (“premise tweet”), and a collection of tweets T_1, T_2, \dots, T_n , we want to rank the collection of tweets from the most similar to the least one to T_p . In order to do, so **we need to develop a method to measure the similarity between pairs of tweets**. Our goal is to use DL to achieve this. By using a DL model trained to rank tweets based on similarity, users could:

- pick a tweet - the example tweet- that they consider useful for detecting conversations about a given disease and
- setup a process to watch the Twitter stream and collect tweets similar to the example one and visualize these tweets sorted by relevance of their similarity. The model can be used to compute a similarity score between the example tweets and each tweet read from the stream. Then, sorting can be used to provide a ranked list based on the similarity.

To accomplish this it was necessary investigate the state-of-the-art methods and techniques related to text analysis, and build a robust architecture using DL algorithms like, CNN and Recurrent Neural Network (RNN) on Natural Language Processing (NLP) approaches, focusing on text similarity measures.

1.3 Key Innovation

Our key innovation is the development of a supervised method to compute similarity between tweets. Given two tweets, T_1 and T_2 , we need to find a metric to compare them. However, establishing this comparison is not trivial. Hence, we developed the following approach. Given a tweet T , that we know has some relevance to a health condition, we call this tweet a premise tweet T_p . We then developed a DL method to compare T_p with two other tweets, T_{H_1} and T_{H_2} , called the hypotheses tweets. This is done with a supervised method based on deep neural networks. Let M' be the model that can determine which of

T_{H_1} and T_{H_2} is more similar to T_p . After training M' on thousands of example triplets of tweets, we can extract a sub-model M out of M' that is capable of scoring the similarity of two tweets T_1 and T_2 on a scale in the real range $r = [1, 4]$. Thus, M does exactly what we need, it scores the similarity between two tweets. Given a tweet T_p we use M to compare T_p to a collection of tweets T_1, T_2, \dots, T_n , and score the similarity between T_p and each T_i . Finally, we can sort this similarity value in decreasing order to rank the collection T_1, T_2, \dots, T_n based on the similarity score between T_p and each T_i .

1.4 Contributions

This thesis represents original contributions by this author to the problem of mining social network data to detect similarity in a collection tweets related with medical conditions, and help build decision support systems for medical applications.

1.4.1 Technical Contributions

Our technical contributions are as follows:

- **Use-case on social networks application to get valuable information about health topics:** Data available on the Internet through social networks, entertainment apps and others, can be used in health-related research. In our case, Twitter is a source of vast amounts of data on different topics that can be transformed into insights about medical issues. We illustrate how we can use tweet data about medical conditions to build models that are able to compute the similarity in tweets. These collection of similar tweets it could be used in the future to support medical applications that try to find trending patterns, anomaly detection, or hidden, repetitive events.
- **Employ Supervised Learning in text similarity tasks:** Most studies about sentence representation (“encoding models”) and text similarity are based on un-

supervised learning (usually clustering) because there is not enough labeled data about a specific task to train a model. This happens in our target problem where we want to find similar tweets related with medical conditions. We show the trained models with labeled data in sentence similarity have good performance to be widely adopted for tweet similarity and in others NLP tasks.

- **Present a novel strategy to build similarity models:** We build models with deep CNN and RNN to obtain acceptable results close to 90% of accuracy. In order to obtain a model M that can compute the similarity rank between two tweets, we borrow from the strategy in [2], and first train a larger model M' that is capable of classifying relative similarity in triplets of tweets. This is a common strategy in DL: *train a bigger model that does a task in which the model you really want is a sub-task (sub-model)*. This is done because the bigger model might be easier to train on its intended task, or because there is data already available to train the bigger model. Once the bigger model is trained, the sub-model is taken out and used in standalone fashion. In our specific case, the task that performs model M , computing the similarity between two tweets, becomes a sub-task in a bigger model M' that classifies which of two tweets is more similar to a premise tweet.
- **Present a performance study on the DL similarity models:** We have created a data sets consisting of 11,425 examples, and we use that data set to train our models. Our results show that we can achieve 90% accuracy on the task of classifying which of two tweets is more similar to a premise tweet. For this task, we need to train a sub-model that computes the similarity rank between two tweets as a value in the range $[0,4]$. This is a regression tasks and our model have a 0.34 units of Mean Squared Error on a decreasing loss.

1.4.2 Contributions to Society

Our contributions to society are as follows:

- Provide a tool that can enable public health officials detect important information about emerging diseases outbreaks .
- Provide data and methods to enable other researchers to improve their efforts in mining social media using deep learning tools.

1.5 Outline

The outline of this thesis is as follows. Chapter 2 contains the literature review on concepts of ML and DL to contextualize our work. In this chapter we also describe topics of NLP and text similarity methods. The problem description and the methodology followed to build the similarity models are described in Chapter 3. In Chapter 4, we explain the ML architecture, implementation, and the different DL models built using CNN and RNN. Chapter 5 shows the results on performance and accuracy of all models built in this work. Finally, Chapter 6 shows the conclusions and future work.

Chapter 2

Literature Review

2.1 Introduction

The field of AI seeks to understand how humans think (“intelligence”), and how to use that knowledge to build programs that show intelligence. Today AI is a very wide field, making it difficult to give a simple answer to what its an AI application. However we can mention a few applications that exist today: autonomous robotic vehicles, speech recognition, autonomous planning and scheduling, game playing, spam fighting, logistics planning, robotics in automation, and machine translation. These applications combine efforts in computer science, engineering, mathematics, and cognitive sciences [1]. These applications need to process a lot of data, hence it is necessary to automate the process of data acquisition and analysis. Big data analytics is used to for the purpose of data management in this context. Machine Learning (ML) (a subfield of AI) provides the tools that automate the process of learning how to extract patterns from the raw data to get insights [3].

Artificial Neural Networks and DL are very popular methods in ML. Neural Networks are simply a collection of connected computational units that represent abstractly the human brain (“neurons”) and aim to achieve learning on a specific task [1]. These neurons are organized and interconnected into layers. Deep learning is the development of complex neural networks that contain many layers of neurons. These networks are computationally expensive to train and require relatively high amounts of training data. But, they can rival human expertise at various taks, such as image classification and language translation.

In this project we are working on similarity tasks related with text messages from Twitter. This problem bear similarity with many applications of similarity for handwritten digits recognition, image similarity detection from text, and image content search in web

searchers (e.g. Google, Bing). Neural Network techniques can be successfully applied for these problems, but to achieve better results and scale to large applications we need used techniques specialized on certain domains, for example in case of NLP (“text analysis”) we need methods to process sequential data, like RNN, that is aDL technique part of ML field [4]. In the rest of this chapter, we go over background information related with machine learning that is necessary to understand the context and the contributions of our work.

2.2 Machine Learning

Machine learning, ML, also known as automated learning, is associated with the concept of making computers learn some tasks without actually programming the computer to do so. Typically, ML involves a training phase, where an algorithm is implemented in software and then presented with many examples of the task to be learned. The training phase produces a *model* which is the trained, “intelligent” software module that automates some tasks. This model is then validated in the second phase, where unseen examples are given to the model and we evaluate its performance. Typically models are evaluated based on either accuracy of prediction (i.e., classification tasks) or on the mean square error incurred when approximating some quantity (i.e., regression tasks). If multiple, competing models are trained and validated, then the final selection is done in a third phase called the *test phase*, where additional examples are given to the competing models to pick the best one amongst them. Notice that this learning is composed of input data that represent previous experience, and the model will learn to produce an output based on such input. [5]. This learning process is focused on gaining knowledge, understanding, experience, and skills [6] in such a manner its performance improves significantly over time.

ML is a very wide field, for this reason it has branched into several sub-fields related to distinct tasks [5] and approaches to solve problems. For the study and application of these algorithms, there are many ways to organize the learning paradigms. The best known are supervised learning and unsupervised learning. This distinction is by what kind of experience they are allowed to have during the training process [4]. Others kinds of ML are semi-supervised learning and reinforcement learning.

2.2.1 Supervised Learning

In Supervised Learning, a model is presented with pairs of input and output items. Often, the input data items in the pairs are called *features*. Likewise, the output data items

in the pairs are often called *labels*. During training, the model automatically learns the relationship between the input features and the labels. In essence, the model is learning a function that maps from the input data to the output data [1, 3]. This kind of learning is only possible if we know the output data [6] and if we have enough examples to train the ML algorithm. A good rule of thumb by Professor Andrew Ng, from Stanford University, is to have at least 10,000 examples to train your models. [7]

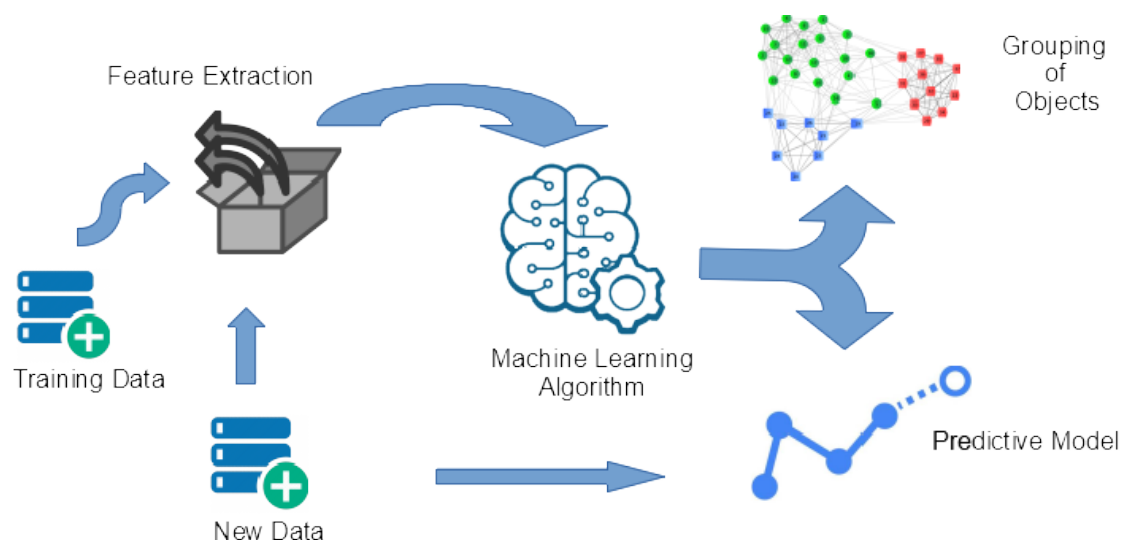


Fig 2.1: Supervised Learning Workflow.

Figure 2.1 shows the process of a supervised learning model. The input of the model is data that contains each element with its respective label. Next the model extracts the most relevant features of the data, to find the relation between input and output, and construct a logical pattern. Finally, the model is trained to learn patterns, generalize data and predict new outputs from new, unseen, unlabeled data items.

The nature of the labels is dependent upon the type of learning problem being tackled. If the supervised tasks involves classification, then the label represents the class or grouping to which the input in the pair belongs. An example classification task will be determining if an image represents a cat or a dog based on the pixels in the image. If the tasks involves regression, then the label represents a quantity associated with the input features. For example, we could predict expected battery life of a lithium battery as a function of daily depth of discharge.

Common techniques used for supervised learning are logistic regression, decision trees, neural networks, and support vector machines. Supervised learning is used in applications such as risk assessment, digital assistants, speech recognition, language translation, image

processing, fraud detection, and customer segmentation [8].

2.2.2 Unsupervised Learning

In unsupervised learning, there are no labels in the data. Instead, the model tries to uncover the underlying structure in the data by finding correlations and identifying patterns parsing the available data [8]. Unsupervised learning is often used as an initial phase during data analysis to figure out the structure of the data. There is no separation of training data and test data [5]; all the data is the input for the algorithm. This method is similar to human behavior: when we observe the world, usually we do inferences and group things based on our interaction with the environment. This type of learning is refined by exposing the model to a lot of observations [8].

The most common technique for this learning modality is called *clustering*, [1], which separates the data into meaningful categories called *clusters*. Each cluster is made up of data items that have similar characteristics [4, 6]. This similarity is measured by using a function that computes a similarity metric, often a variation of a distance function. In practice, the items are represented as vectors v_1, v_2, \dots, v_k and the metrics measures how close are to each other in their vector space. Other unsupervised techniques are nearest neighbor mapping, singular value decomposition, self-organizing maps, Hidden Markov Models, and anomaly detection, to name a few. Their applications are related to market basket analysis, anomaly/intrusion detection, text classification, identifying similarities amongst data elements, sentiment analysis, and so on [8, 9].

In Figure 2.2 we show a workflow for unsupervised learning. In this case, we show clustering of documents based on content. First, the text and other features in documents are converted into a vector representation (i.e., one-hot vectors). Next the model extracts the most relevant features of the data, then a clustering algorithm (e.g., k-means clustering) is used to assign each vector to a cluster. The number of cluster is pre-defined at the start of the learning process. Once trained, the model is able to take new data items and cluster them.

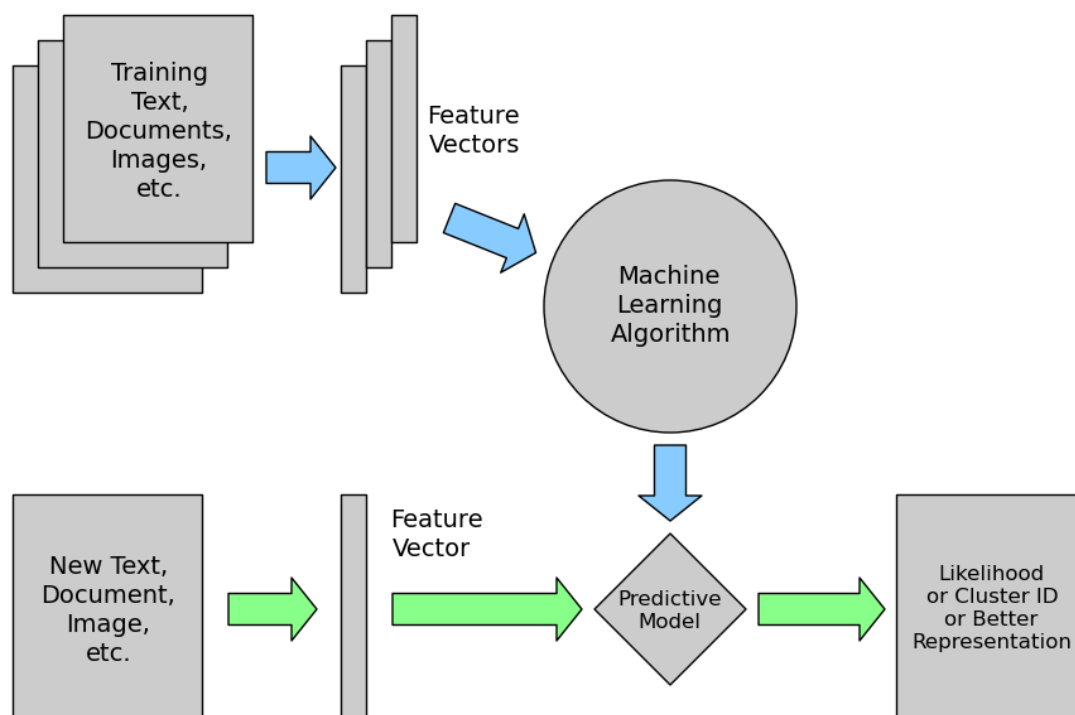


Fig 2.2: Unsupervised Learning Workflow.

2.3 Neural Networks

An artificial neural network is a very simplified abstract model of a biological brain, an interconnection of processing units with capability to learn patterns, generalize and associate data. A significant aspect adopted from biological brain is the “learning capability” from the experience and transfer knowledge to find reasonable solutions to similar tasks [10, 11].

When we speak about a neural network, it can be something as simple as a network with a single node or more complicated as a collection of nodes stacked in layers[11]. The structure of the neural network basically consists of: 1) a set of linked nodes associated with a numeric weight that represent the strength of connection between them, 2) an input function for each node that computes the weighted sum of all inputs, and 3) an *activation function* to control the neuron “trigger” behavior and get the desired output [1, 11] .

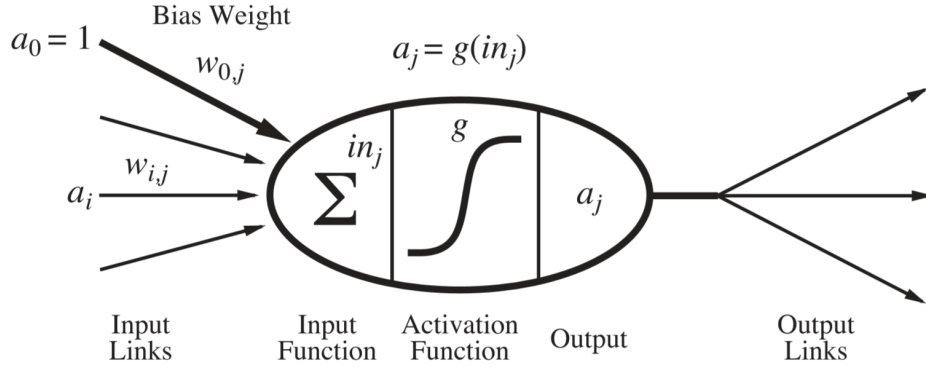


Fig 2.3: A Simple Mathematical Neuron Representation as depicted in [1].

Figure 2.3 shows the basic structure of a simple mathematical neuron, where the output is the result of applying an activation function to the weighted sum of inputs. The input is a collection values of a_1, a_2, \dots, a_i . The output is a value a_j . Each of the inputs are multiplied by a weight $w_{i,j}$ which is a real number. The weights are important to minimize the cost of activation function and they are updated when the model is trained, through an optimization process. Often an additional term $a_0 = 1$ and weight $w_{0,j} = 1$ are added as the bias term in the equation. Equation 2.1 represent the output after applying an activation function in the neuron showed in Figure 2.3.

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

The activation function g provides a way to add a non-linear relationship between the inputs and the output. Some of the functions used as activation functions are the sigmoid function, the hyperbolic tangent function, and the rectified linear unit function (RELU). [4].

An alternative formulation, based on linear algebra, represents the examples as a collection of vectors X_1, X_2, \dots, X_k . These vectors are stacked horizontally on to form a matrix X . Next, we have the labels Y_1, Y_2, \dots, Y_k which are also stacked in a matrix Y . Then the equation for the neural network becomes:

$$Z = WX + b \quad (2.2)$$

$$Y = g(Z) \quad (2.3)$$

In this formulation, WX is the matrix multiplication between the examples and the weights and b is the bias vector. The variable Z represents the result of these operations and becomes the input to the activation function g which finally yields the output of the neuron represented by the variable Y .

2.3.1 Sigmoid function

The *sigmoid function* is used to compute binary activation values in the range $[0, 1]$. The function is given by the following equation:

$$\sigma(Z) = \frac{1}{1 + e^{-Z}} \quad (2.4)$$

The sigmoid function is interpreted as computing a probability and it is used in binary classification. Typically, an input on which the sigmoid gives a values greater or equal to 0.5 is classified to the 1 class, whereas a values less than 0.5 is sent to the 0 class. The sigmoid function is typically used in the final node of a neural network.

2.3.2 Hyperbolic tangent function

The *hyperbolic tangent function* is used to compute an activation in the range $[-1, 1]$. and is used for activation in intermediate nodes in a network. The function is given by the following equation:

$$\tanh(Z) = \frac{e^Z - e^{-Z}}{e^Z + e^{-Z}} \quad (2.5)$$

Typically, \tanh is used for activation in intermediate nodes in a network.

2.3.3 Rectified linear unit function

The *Rectified Linear Unit (ReLU)* is used to compute an activation in the range $[0, z]$, where z is a real number. The function is given by the following equation:

$$g(Z) = \max(0, Z) \quad (2.6)$$

Typically, ReLU is used for activation in intermediate nodes in a network. This function provides better convergence in deep neural networks.

2.3.4 Softmax function

The *softmax function* is used to map a vector K with n dimension into n probability values. The formula is:

$$s(K_i) = \frac{e^{K_i}}{\sum_{j=1}^n e^{K_j}} \quad (2.7)$$

Softmax is always used in the final node of a neural network used for multi-class classification. The value $s(K_i)$ represents the probability that the input example belongs to class i . Typically, the max value of all K_i is used to select the class i with highest probability.

2.3.5 Connecting Layers

To form a network, we need to connect individual neurons as nodes in a network graph. There are many ways to build the network, but feed-forward networks are the most common ones. Figure 2.4 shows the topology of a feed-forward network. In this organization, the components are clearly separated: an input layer, an output layer and one or more hidden layers (also called processing layers). The input data is fed at the input layer, and the output label is generated by the output layer. For a given neuron at layer l , its output is fed as input to neurons in the next layer $l + 1$.

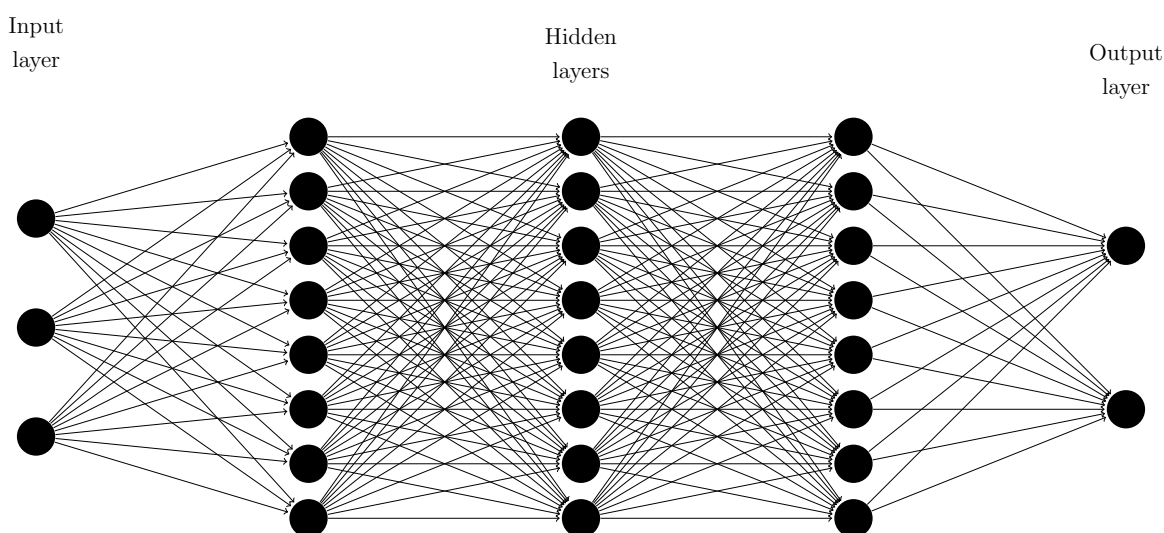


Fig 2.4: Feed Forward Network

2.3.6 Parameters and Hyperparameters

The *parameters* of a neural network model are the weights $w_{i,j}$ in each layer. Often, the notation $W^{[l]}$ is used to denote the weight matrix W used at layer l . If we have a network with n layers, then we will have n weight matrices $W^{[1]}, W^{[2]}, \dots, W^{[l]}, \dots, W^{[n]}$. Again, the values in these matrices are called the *parameters* of the model. The ML training process involves the use of an optimization algorithm that finds the right set of parameters $W^{[l]}$ for each layer l . These parameters are chosen as to minimize the error between the predicted label $\hat{Y}^{(i)}$ and the actual label $Y^{(i)}$ for each input example $X^{(i)}$. The most popular algorithm in use is called Gradient Descent, and has many variations such as ADAM, RMSprop, and ADADELTA [4].

In most instances, there might be multiple models to use on a given problem. In addition, multiple optimization algorithms can be tried to train a particular model. These different alternatives have an impact on the training process and the parameters of the models. They are often called *hyperparameters* because they are not really parameters of the model, but are variables that affect training. Example hyperparameters include: a) network architecture, b) optimization algorithms, c) learning rate for the optimization process, and d) methods to prevent overfitting. *Overfitting* is a situation where the model works well on the training data but fails to generalize to the validation data set.

2.3.7 Data Splitting

The recommended procedure to find the right model involves splitting the sample data into three parts:

1. **Training Dataset** - this is the data used to train the network and find the parameters of the model. It is recommended to use 60% of the data for this task.
2. **Validation Dataset** - this data is used to validate the performance of the model for a given combination of hyperparameters used for training. It is recommended to use 20% of the data for this task.
3. **Test Dataset** - this data is used to select the best model from a collection of models that have been trained and validated. It is recommended to use 20% of the data for this task.

2.4 Deep Learning with Neural Networks

Traditional neural networks only contain a few layers, for example: one input layer, one to three hidden layers, and a final output layer. Modern approaches, however, add dozens of hidden layers with tens or even hundreds of neurons in each layer. These approaches result in models that form complex and deep networks graphs, much more powerful than a regular neural network. For that reason, we know this approach as “DL” [4, 8, 12]. Improvements in hardware, particularly Graphical Processing Units (GPU) units for massive parallel computing of neural networks, have enable this “deep approach”. Without this technology, the computational costs to find the parameters in the deep networks would made them difficult, if not impossible, to train because they contain millions of parameters. In contrast, small neural networks with just a couple of layers have just a few thousand parameters and are easier to train.

The most typical example of DL approach is based on the feed-forward deep network. It consists in an input layer, that contains the input data for the model, next we have the hidden layers, a variable number of neurons and layers, and finally an output layer as is showed in the Figure 2.4. By increasing the depth (number of hidden layers) on the network, it becomes possible to learn more complicated relationships. Other important methods for DL include the convolutional neural network (CNN) which are used when working with images, and the recurrent neural network (RNN), used to process sequence data.

2.4.1 Fully Connected Networks

Fully connected networks are the traditional feed-forward networks with many layers, as shown in Figure 2.4. Recall that for each node at layer l , its output is connected as input to all nodes in the next layer $l + 1$.

2.4.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are applied to problems with multi-dimensional data that can be represented in tensor forms (e.g. 2-D matrices, or 3-D cubes). For simplicity, the input of a CNN is called an image since they are extensively used for image processing. The key idea in these networks is to scan the input image and apply the *convolution* operation between portions of the image and one *filter*. A filter is a smaller matrix W

which contains the parameters for the model. The operation is expressed as:

$$Z = W * X + b \quad (2.8)$$

where X is the image, W is the filter, b is the bias vector, and $*$ is the convolution operation. Like in the case of fully-connected layers, an activation function g can be applied to the vector Z . In convolutional layers, ReLU and *tanh* are often used as activation functions.

Each layer in a CNN is focused in applying a convolution operation. There might be more than one filter in each layer, and there can be additional “pooling layers” used to sub-sample the image and reduce its size. Current DL approaches stack multiple convolutional layers in a row and then flatten the image to a one dimensional array. This array is then feed into a sequence of fully connected layers that end with an output layer.

CNNs are extensively used in image processing and pattern recognition [2], as a powerful visual model to extract features in a hierarchy of concepts [13]. Historically this type of network was some the firsts to solve important commercial problems [4], like the handwritten zip code number recognition.

2.4.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are networks for processing sequential data. Examples include speech recognition, language translation, and financial time series data. RNNs are very powerful because they efficiently store information about the past. The basic idea in a RNN is to have a layer of neurons that process an input data item and produces an output based on the input and a previous value from previous step. Thus, the network forms a feedback loop where the output in one time step t is used in combination with the input at time step $t + 1$ to produce the output for step $t + 1$. In essence, the RNN has a memory of previous events when it produces the next output value. Like in the case of CNN, the final layers of the RNN are fully-connected layers used to produce the output of the model.

RNN are specialized to process a sequence of values [4], mapping all the previous inputs to the final output. This allows the memory of old inputs to persist and influence in the next network output [14]. RNNs have also been used in image classification [12]. Other applications are found in the text processing: sentiment classification, topic classification, summarization, and machine translation.

2.4.3.1 RNN based on Long Short-Term Memory (LSTM)

This type of network manages the contextual information with the ability of “forgetting” the information that is not useful and maintaining the information that has more probability to be used in next processes. Each layer has the ability to forget some previous state and give more weight to new input values. To control the flow of information over the gates of the network, there are specialized neural units to control how the relative weight of the current vs the output of the previous step [15]. There is a forget gate that controls when to forget previous information and another gate to control how much importance does the new information has.

2.4.3.2 RNN based on Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is an easier way to implement a recurrent network with control of memory, and it was is meant to be is a special case of a Long Short-Term Memory (LSTM). In a GRU, there is only one gate that controls both the ability to forget about previous data and give more importance to new input values [15].

2.5 Natural Language Processing

Since our goal is to find a way to measure the similarity between text in tweets, we need to briefly discuss the field of natural language processing (NLP).

Basically, NLP is the process to understand, analyze and use the human language by machines or algorithms. This includes written text, voice commands, or both. The goal is to allow computers to comprehend natural language, receive commands, and produce results in human language. There is a lot of information contained in web pages, video sites, and social networks. Almost all of it is written or recorded in natural language, and that is a major reason why we want our algorithms to understand human language, in order to acquire all this information from written and spoken language. A ML model that wants to acquire knowledge needs to comprehend at least partially the use of human language (e.g. ambiguity and messiness). This undertaking is not trivial because the presence of the colloquialism, figure of speech, abbreviations and emoticons, to mention a few [1, 8]. The difference with a computer language (e.g. python or java) is that a natural language cannot be characterized in a set of sentences, because they are very large, ambiguous, and constantly changing; in the state-of-the-art models models are just an approximation [1].

The process of language analysis is closely related to linguistic, traditionally, this

process is decomposed in three stages: syntax, semantic and pragmatics. We first analyze the syntax of a text providing an order and a structure, then analyze the text in terms of meaning (semantic), and the last stage is a pragmatic analysis, which relates the sentence with a context. Such separations constitute the basis for structural models from a software point of view, making the analysis more manageable and serving as a starting point. Recently, deep learning solutions are not focused on this scheme and they try to understand language straight from the raw data examples [16].

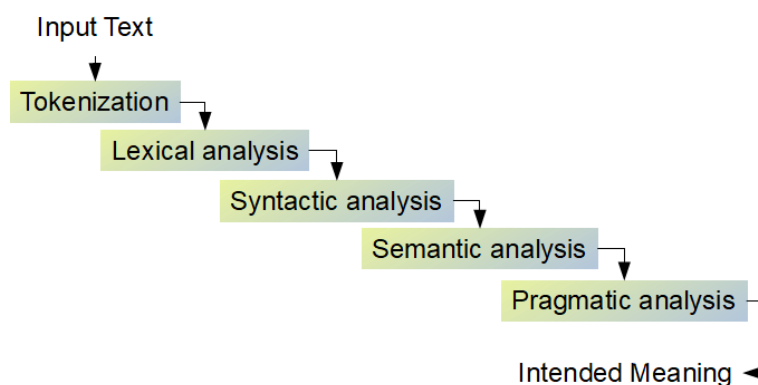


Fig 2.5: Natural Language Process Stages

Figure 2.5 represent the stages in a traditional NLP pipeline. The first step, tokenization, divides a continuous text into words or sentences. In English, text is usually separated by whitespaces but other unsegmented languages are more difficult because there is not a visible separation between words (e.g. Chinese, Japanese). The second stage is lexical analysis, which refers to morphological processing, mapping strings to lemma and assigning structure to words, and registering their structural properties. The third stage is syntactic analysis where a sentence is processed to determine its structural composition, following rules of a formal grammar to finally assign a meaning to the sentence. The syntax tree is the standard to represent a syntactic structure which contains the steps in the derivation from the initial sentence. The fourth stage is semantic analysis, and it is related with the context of words, phrases, etc. This stage not only refers to the meaning of a sentence, but also to its relationship with other words or sentences, to infer meaning of the text. The final stage is pragmatic analysis, also called natural language generation, and it is the process when a machine can understand natural language as input and produce natural language as output with the intention to communicate something with a correct context [16].

Applications of NLP are based on probability distributions over sequential data in

a natural language. We can use generic neural networks for basic tasks, however, in complex applications, we need specialized techniques to process data, sometimes like a sequence of words, characters or even bytes. The earliest successful language model, a probability distribution over sequential data, was n-gram models (sequences of tokens). It has many possible types: unigrams, bigram, trigram, n-gram. In order to improve statistical efficiency, researchers introduced the notion of word categories or class-based n-grams models; however these models are not able to share statistical strength of similar words and their contexts.

Neural network models have been able to recognize the similarity not only between words but also in their contexts and how words are distributed with each other. This word representation sometimes is called distributed representation or word embedding, which means that language symbols are represented as points in a multi-dimensional vector space. In an embedding space, words with similar meaning are close to each other [1, 4] because they are represented by vectors that are close by in the vector space (in the Euclidean distance sense). The best known word representation are Word2Vec Models [17], an improved of skip-gram model in terms of quality of vectors, and speed training [17] and GloVe model that combines the advantages of global matrix factorization and local context window [18]. More recent studies show other sentences encoder like Skip-Thought Vectors, an approach for unsupervised learning that tries to reconstruct passages of text that share semantic properties [19]. There is also the Supervised Learning of Universal Sentence Representations from Natural Language Inference Data, an encoder using pre-trained sentence level embedding with supervised data of Stanford Natural Language Inference datasets demonstrating stronger transfer task than skip-thought vectors [20]. Also, we have the Universal Sentence Encoder, which uses supervised data to train a sentence embedding model instead of a word level embedding, showing a good performance [21].

There are several important applications of DL techniques in NLP tasks including:

- **Spelling and Grammar Checking:** The suggestions that appear when we type a text in a smartphone or when we redact a document in an advance text editor.
- **Text Classification:** It is the categorization of a text in a defined set of classes (e.g., hate speech), or the language identification, or the spam detection in email messages [1].
- **Information Retrieval:** Search, retrieval, and ranking of documents responding to a keyword query.
- **Summarization:** Automatic generation of the essential information from a long document into a shorter piece of text.

- **Syntactic Analysis:** The analysis of a string of words to isolate subjects, verbs, adverbs, adjectives, and others complements (phrase structure) [1].
- **Machine Translation:** It is the task to automatic translation of a sentence into an equivalent sentence in meaning in another natural language [4].
- **Speech Recognition:** This task identifies the words spoken to determine their meaning. That is difficult because the ambiguity and noise are a hard challenge [1].

2.6 Text Similarity

Text similarity refers to how close is a piece of text T_1 to another one T_2 in terms of meaning and structure. The first means semantic similarity and the last is called lexical similarity. All works in lexical similarity often are developed to achieve semantic similarity [22]. Lexical similarity exists when there is character or word matching (e.g. “feel” and “feet”). On other hand, semantic similarity is based on meaning and context (e.g. “President of the United States’ and “POTUS”) [23]. Text similarity is used in many NLP problems such as text classification, clustering, information retrieval, topic detection, machine translation, summarization and others [23, 24], [25]. It is used heavily in research related with social network analysis [26].

2.6.1 Lexical Similarity

Lexical similarity refers to structurally similar words or characters in the evaluation sentences (matching or comparison). Lexical similarity could categorize text depending of the granularity used. It can be character level, word level, or phrase level [22, 23]. String-based methods to calculate the measure of similarity are categorized as follows:

- **Character based similarity:** There are many techniques based on characters. The longest common substring/subsequence is a method that finds substrings and compares them based on a common chain of characters they both share. This is the longest common chain of suffixes for each substring. Another idea is the Levenshtein distance, where the similarity measure is given by the number of operations to transform one string in another one, using insertions, deletion or substitution of adjacent characters. The Jaro distance is another method that emphasizes in the number and order of matching common characters; it is used in duplicate detection (record linkage). Finally, the n-gram method calculates the distance of two sub sequences, dividing its similar n-grams by maximal number of n-grams. [23, 24, 25].

- **Statement/Term based similarity:** A method widely known in text similarity is cosine similarity, where the similarity distance is given by the cosine of the angle between two vectors in an inner product space. These vectors represent the sentences being considered. Euclidean distance, also called L2 distance, use vectors in a Euclidean space where the distance is the square root of the sum of squared difference between two vectors (sentences). Jaccard similarity coefficient or Jaccard index is used to calculate similarity and diversity in sets; it takes the shared term in the intersection and divides it by the number of all terms of the union. There are other known methods like block distance (Manhattan distance, L1 distance, etc.), Dice's Coefficient, Matching coefficient, overlap coefficient, soft-cosine similarity and centroid based similarity, among others [23, 24, 25].

2.6.2 Semantic Similarity

Semantic analysis entails a deeper level of analysis, including the meaning and the context of sentences. For example syntactic parsing to get dependency structure in the phrases. Commonly semantic similarity use background information about concepts like WordNet, or Wikipedia, or a simplify corpora of texts [22, 26]. Semantic similarity is frequently used for text summarization, topic analysis, recommendation system, and collaborative tagging system, among others. This approach could be classified in three big categories based in the use of information: corpus based, knowledge based and hybrid methods [25, 26], described below.

- **Corpus based similarity:** Also called statistical similarity, this approach refers to the use of information gathered from a large corpus from written o spoken data. Latent semantic analysis represents the text in a matrix where rows are the unique words and columns represent each sentence. This method is constructed based in a corpus of text and a mathematical technique called singular value decomposition to reduce dimensionality while maintaining similarity relation between word and text. To calculate the similarity is often used cosine similarity. Another technique is hyperspace analogue to language, a variation of latent semantic analysis, taken a set of words called a “window”, and is compared with the corpus to calculate co-occurrences in words, forming a matrix representing the strength between related words. Its similarity also can measure with cosine similarity method. Explicit semantic analysis uses the Wikipedia corpus to convert sentences in a tf-idf (term-frequency-inverse document frequency) weighted vector between each word and the documents of corpus. The measure to calculate the distance between vector is cosine

similarity. Normalize Google distance is a semantic metric that uses Google search to get the number of hits returned for each term of the text and build a metric to calculate the words with similar meaning between two sentences. Other methods are point wise mutual information, normalized information distance, and normalize compression distance [22, 24, 25, 26].

- **Knowledge based similarity:** It is called topological similarity and it is based in semantic networks; these networks are a lexical database grouped in set of semantic synonyms with conceptual and lexical relations that include verbs, nouns, adverbs, and adjectives. The most popular semantic networks are Word Net and Natural Language Toolkit [23, 24, 25]. There are distinct types of categorization focused in similarity or relatedness. The node-based (also called information content based) similarity that calculates the similarity between two sentences by the amount of information that share each other; Resnik similarity method uses only Word Net nouns to get the information content of sentences; other measures are Lin method, Jiang and Conrath method [23, 24, 26]. Edge-based approach counts the edges of graphs (semantic network) between the compared concept nodes where nodes with shorter path are more similar. All edges are weighted considering network density, node depth, type of link, and link strength [24, 26]. Other approach is feature-based, which uses the concepts of Word Net as a list of features to calculate the semantic similarity between sentences [26], and finally gloss-based approach uses glosses of words from a given corpus; Vector Measure is a method that forms a co-occurrence matrix with the average of each co-occurrences vector of a gloss/concept to measure the semantic similarity [23, 26].
- **Hybrid similarity measures:** This approach is a combination of many features, like lexical structure and corpus information, combining several metrics into one. For example, semantic text similarity method is a combination of syntactic and semantic information [25, 26]. Other methods combine more approaches, according to [27] which built a supervised random forest regression model that use machine learning methods to combine string features, corpus-based methods, knowledge-based methods, syntactic features, and multi-level text features.

Chapter 3

Problem Formulation

3.1 Description

Our work is rooted in THS project at UPRM, whose goal is to monitor tweets related to medical conditions, and detect instances where a disease is actively being discussed in a given region. The main problem that we want to tackle in this project is the development of DL models that can evaluate the similarity between tweets. Specifically, given an example tweet T_p and a collection of tweets T_1, T_2, \dots, T_n , we want to rank the collection of tweets from the most similar to the least one to T_p . In order to do, **we need to develop a method to measure the similarity between pairs of tweets**. Our goal is to use DL to achieve this.

First, we need to create a collection of labeled tweets to help us determine how humans view similarity. In this collection, we have rows with triplets (T_p, T_{h_1}, T_{h_2}) , each element being a tweet. For each row, the first tweet, T_p is the “example” tweet called the “premise”. Next to it, we have two additional tweets, T_{h_1}, T_{h_2} , called the hypothesis tweets. Users are asked to compare the premise with each of the two hypothesis tweets, and rank which one is more similar to the premise. This is done by providing a numeric value on their similarity. The value is in the range (0,4), with 0 being unrelated tweets and 4 being identical in term of meaning. Thus, for example, if the rank between T_p and T_{h_1} is 1, and the rank between T_p and T_{h_2} is 3, then tweet T_{h_2} is more similar to T_p .

With this labeled data set, we then aim to train a model than can learn to do this ranking and learn how to give a similarity score between the example tweet T_p and another tweet T_i that we want to consider. Our approach borrows ideas from the work in [2] which find similarity between images.

By using a DL model trained to rank tweets based on similarity, users could:

- pick a tweet - the example tweet- that they consider useful for detecting conversations about a given disease and
- setup a process to watch the Twitter stream, collect tweets similar to the example one, and visualize these tweets sorted by relevance of their similarity. The model can be used to compute a similarity score between the example tweets and each tweet read from the stream. Then, sorting can be used to provide a ranked list based on the similarity.

As we described in section 2.6.2, text similarity measures are very difficult to calculate because it is hard to find the meaning and context match of the two pieces of text. To manage those issues, we use relevant information to find context similarity and syntactical similarity, like the type of disease mentioned in each tweet, or if the tweet is related to a disease or not. This contextual information can be used as pre-processing step before feeding the tweets to the DL model.

3.1.1 Examples

We now provide some examples of tweets to show the complexity of working with data related to semantic measures and contextual similarity. We obtained these tweets from the actual Twitter stream.

Example 1 : Tweets that are not related and discuss different diseases.

- *Measles starts with cold like symptoms that develop about of days after becoming infected this is followed a few days later by the measles rash poster shows us the symptoms to look out for think measles prevention is best a dose of vaccine needed vaccines work.*
- *The Ebola outbreak was a worst nightmare for countries affected interestingly Ebola was not a new disease in fact on managing the disease there is a forty year old knowledge so how did Ebola become such an epidemic.*

In this example, the two tweets discuss about diseases. The first one of them describes the symptoms of measles, and recommends vaccines as the method for prevention. The second tweet does a question about the reason why Ebola has expanded if this disease is not new and there is enough knowledge about it. The content of these two tweets are related with a health condition but they are describing different diseases, that is enough to consider them not similar.

Example 2 : Tweets that are not related and discuss the same disease

- *Theory man flu exists as a phenomenon because it is the only time toxic masculinity tells men it is okay to feel vulnerable weak and look to their partners for support.*
- *Yeah, I missed like a day of school on and off flu I could not even eat, and I could not walk without feeling like vomiting or collapsing.*

The first tweet refers to the flu, but it is not about a health condition. It discuss an assumption about how the flu exists to show the vulnerability of weak men. The other tweet is discussing about someone that missed a school day because flu and describes his/her symptoms.

Example 3 : Tweets that are related and discuss the same disease

- *I wish I knew if the twins were going to get the flu like what is the usual window, he is had it symptom wise since Sunday night and it is Tuesday so when would they likely show symptoms ahh.*
- *So I was at the hospital and obviously could not type while I was there since, a I could not type and b I was scared of being caught with lewd rps after I healed bad luck struck again and I was bedridden with the flu all I did was sleep and I was coughing so much my eyes went red.*

In this case the first tweet mentions to twins getting flu and describe the intervals of days with possible symptoms. The second tweet is about a person who was in the hospital and explains that was the reason because he could not type, and when he was recuperated, then he got flu again and just goes to sleep. The two sentences are showing a health condition (flu) where the writers are explaining symptoms and what they were doing when they got flu. Therefore, these tweets are related.

3.2 Formalization

We are now in a position to better formalize the problem at hand, considering a premise tweet T_p and a collection of tweets T_1, T_2, \dots, T_n . Let $diff(T_i, T_j)$ be a function that computes the similarity between two tweets T_i and T_j as their distance in some n-dimensional embedding (vector) space. The smaller the value for $diff()$, the more similar are T_i and T_j . Let $rank(T_i, T_j)$ be a function that assigns a comparison rank to T_i and T_j in the range (0,4); the closer to the value 4, the more similar are T_i and T_j . We define the following property for $rank(T_i, T_j)$:

Property 1 Rank Property

Given three tweets T_p, T_{h_1}, T_{h_2} , if $rank(T_p, T_{h_1}) > rank(T_p, T_{h_2})$ then $diff(T_p, T_{h_1}) < diff(T_p, T_{h_2})$.

With this property we establish the $rank()$ as an alternative method to measure similarity. The higher the rank the more similar the tweets. Our goal is then train a model M to compute the function $rank(T_i, T_j)$.

Notice that the $diff()$ function is an unbiased measure of similarity, but it is difficult to find a general similarity metric, as discussed in Chapter 2. Instead, our $rank()$ function tries to approximate human ability to make the comparison and establish similarity.

3.2.1 Scope of the Model

ML models are bound to work on data coming from the same distribution as the data for training. In order to properly scope our model, we limit the data set to include tweets that contain the following medical keywords:

- Flu
- Zika
- Ebola
- Measles
- Diarrhea

Sometimes, the keywords mentioned above are used in colloquial conversations that are not related with medical issues, which can mislead researchers into believing the occurrence of a keyword implies a disease outbreak. For that reason, the tweets are then organized and labeled in three classes:

- **0**: Tweet does not belong to a medical condition.
- **1**: Tweet belong to a medical condition.
- **2**: Tweet has an ambiguous meaning, it is difficult to classify

Clearly, in a production setting, we are only interested in working with tweets that belong to class 1.

3.2.2 Use in Production

Once the model M is trained, we want to use it in the production task following these steps:

1. Subscribe to the Twitter data stream to build a collection $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$.

2. Chose a premise tweet T_p .
3. Compute the $rank(T_p, T_i)$ for each tweet $T_i \in \mathbf{T}$, and store the triplet $(T_p, T_i, rank(T_p, T_i))$.
4. Sort the triplets $(T_p, T_i, rank(T_p, T_i))$ in decreasing rank order.

With this procedure, the tweets more similar to T_p are sorted from most similar to least similar.

3.3 Data Processing Architecture

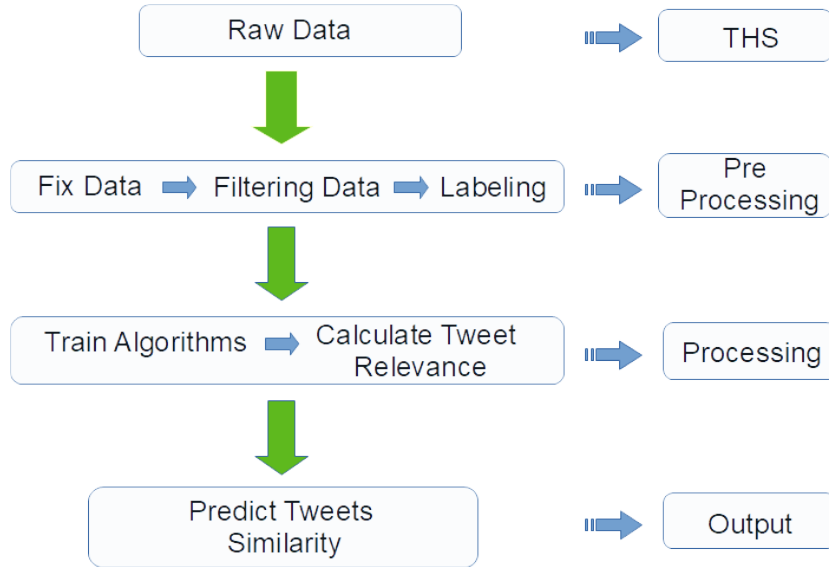


Fig 3.1: Data Processing Architecture

Figure 3.1 depicts the architecture for tweet processing that we shall implement to train and use in our models. The architecture is based in the THS system [28, 29]. Tweets are collected in the THS system and stored in a distributed database system (Hive) using Hadoop Distributed File System (HDFS). Tweets are “cleaned”, removing links, hashtags and mentions. In the next stage, tweets are filtered and associated to a disease. After detecting if the text is related to a disease, tweets are labeled to determine if each tweet is discussing about a disease or if it is a lexical or semantical ambiguity. This is done using the model trained in [29]. In order to train the similarity model, triplets of tweets are given to human labelers so they can provide a score of similarity between the first tweet (premise) and the other two tweets (the hypotheses). Once the similarity model is trained, it can be used by the production model for find the similarity of tweets.

Chapter 4

System Architecture and Organization

4.1 THS System Overview

THS is a collection of Big Data tools running on a cluster system at UPRM to collect, filter, and store the tweet data to future uses. We use ML tools to process data; such as Keras [30], TensorFlow [31], Scikit-learn, and others. Our current THS system consists of 12 nodes in a big data environment, organized as follows: One master-node, one client-node, and 10 data-nodes. The big data tools installed are HDFS [32], Yet Another Resource Negotiator (YARN) [32], Apache Kafka [33], Apache Spark [34], and Apache Hive [35].

The data processing pipeline is shown in Figure 4.1 and detailed step by step as follows:

- **Step 1:** We subscribe to the Twitter Application Program Interface (API) to save raw tweet data, building a real-time data stream using Kafka, a distributed streaming platform [33]. The script (“producer.py”) runs on the edge-node (client-node) of our cluster identified as node05.ece.uprm.edu. In this step, we filter data by language (e.g. English) and by keywords of target medical diseases (e.g. Flu, Measles). The purpose of this script is to keep with the Twitter stream, collecting data, and passing it down the pipeline for its eventual processing.
- **Step 2:** All raw data obtained from the Twitter API is stored in the Kafka pipeline for later processing. Each stored record consists of a key, the data, and a timestamp. Kafka is configured to keep data for 3 hours; after that time, the system discards raw data automatically to save disk space.

- **Step 3:** In another script called “consumer.py” we connect to a stream “topic” in Kafka to get tweets by arrival order. This script is in turn connected to the Apache Spark streaming system. The purpose of this script is to remove the data from the Kafka queue for processing. Thus, the producer and consumer scripts implement an asynchronous data pulling pipeline.
- **Step 4:** The Spark streaming system collects the data from the consumer and begins to clean data, filtering and keeping only real tweets, and discarding re-tweets or replies.
- **Step 5:** Data is saved in Hive tables at regular time intervals with specific information that include users, keywords, hashtags, geolocation, and the original raw tweet. This data is used to future processes and searches.
- **Step 6:** Data stored in Hive database is used to generate data visualizations and, in this project, to get data samples to train a neural network models to measure the similarity between tweets.

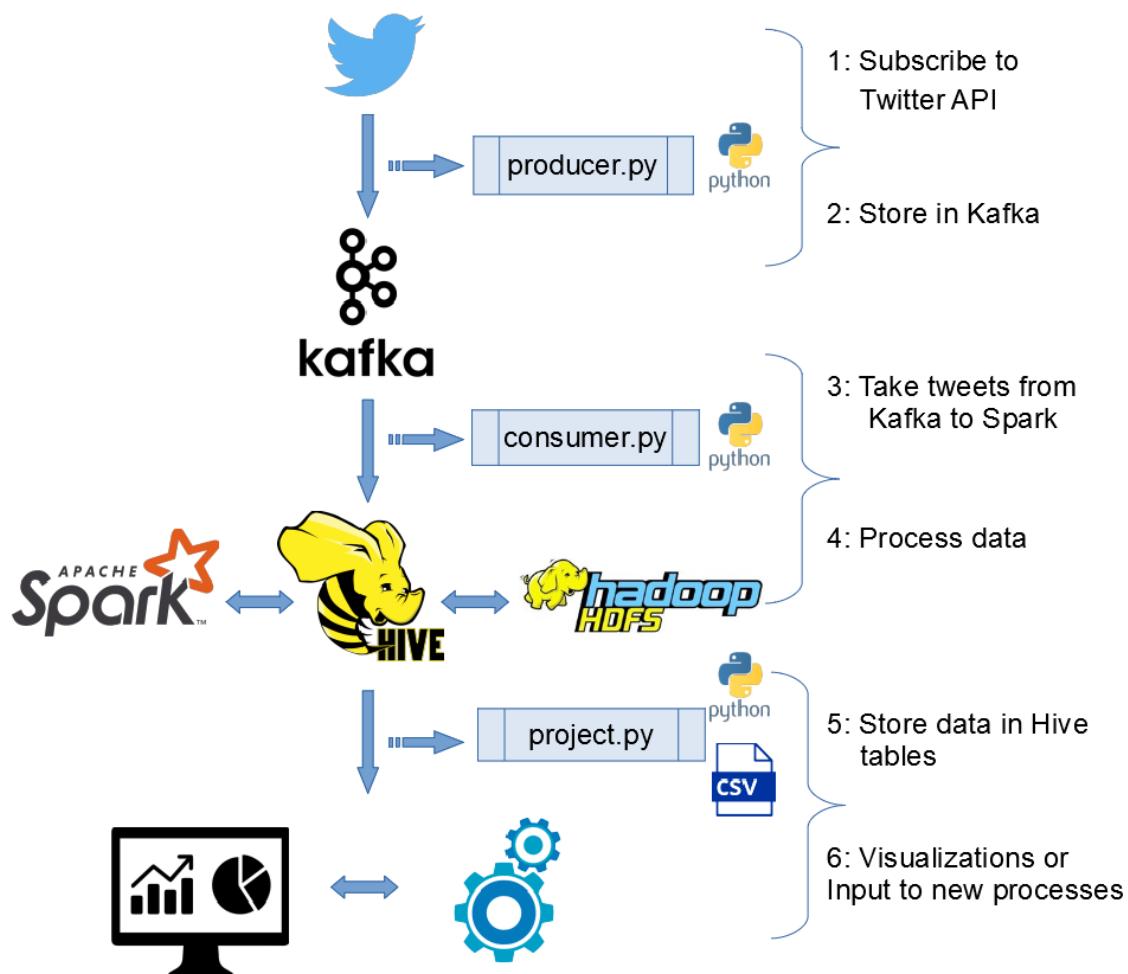


Fig 4.1: THS Processing Architecture

4.2 Strategy to Obtain the Model for Similarity

In order to obtain a model M that can compute the similarity rank between two tweets, we follow the strategy in [2], and first train a bigger model M' that is capable of classifying relative similarity in triplets of tweets. This is a common strategy in DL: *train a bigger model that does a task in which the model you really want is a sub-task (sub-model)*. This is done because the bigger model might be easier to train on its intended task, or because there is data already available to train the bigger model. Once the bigger model is trained, the sub-model is taken out and used in standalone fashion. In our specific case, the task that performs model M , computing the similarity between two tweets, becomes a sub-task in a bigger model M' . In this case, M' receives a triplet (T_p, T_{h_1}, T_{h_2}) and emits

a classification value as follows:

- 0 - tweet T_{h_1} is more similar to T_p than is tweet T_{h_2} .
- 1 - tweet T_{h_2} is more similar to T_p than is tweet T_{h_1} .

In order to train M' , we need to first compute the similarity between pairs of tweets, and this is exactly what model M will do. But it is much easier to ask humans to establish a rank between three tweets, as opposed to ranking many more tweets and trying to compare just two of them. The authors in [2] used this same idea but for the case of computing image similarity. Our approach is to make M part of M' , train M' , and then extract the sub-components in M' that belong to M .

4.3 Labeling

In order to work with the strategy presented in section 4.2, we created a data set for labeling. The dataset consisted of 4,225 triplets (T_p, T_{h_1}, T_{h_2}) . All the tweets in these triplets were known to be related with a medical conditions, and had one of the five target keywords: Ebola, Zika, Flu, Measles, or Diarrhea. There were 54 labelers, which were students from UPRM that were enrolled in the course “Introduction to Database Systems”. Labelers were instructed to:

- compare T_p with T_{h_1} and T_{h_2} ,
- rank the similarity between T_p and T_{h_1} on a scale in the range (1,4), and
- rank the similarity between T_p and T_{h_2} on a scale in the range (1,4).

4.4 Building the Dataset for Training

In all of our data sets, the first tweet in the triplet, T_p , is always related with a disease. We took the data set from the previous section, and augmented it by adding two additional group of triplets. In the first group, one of the hypothesis tweet, T_{h_i} has the same disease keyword as T_p but is not related with a medical condition (class 0). In this case, the ranking of this tweet with respect to T_p is set to 0. In the second group, one of the hypothesis tweet, T_{h_i} has a different disease keyword as that T_p . In this case, the ranking of this tweet with respect to T_p is also set to 0. In total, we grew the training data set to 11,425 tweets.

The next step was to prepare the data and align it according to the model that we wanted to train. We took every triplet (T_p, T_{h_1}, T_{h_2}) and converted into corresponding tuples (a finite ordered list of elements) of the following shape:

$$(T_p, D_{T_p}, M_{T_p}, T_{h_1}, D_{T_{h_1}}, M_{T_{h_1}}, T_{h_2}, D_{T_{h_2}}, M_{T_{h_2}}, \text{rank}(T_p, T_{h_1}), \text{rank}(T_p, T_{h_2}), L)$$

The structure of these tuples is as follows:

- T_p - premise tweet.
- D_{T_p} - index for the medical condition in T_p . Indices used: 0 - Flu , 1 - Ebola , 2 - Measles , 3 - Diarrhea, 4 - Zika.
- M_{T_p} - classification label of the tweet T_p : 0 - not related with a medical condition, 1 - related with a medical condition, 2 - ambiguous.
- T_{h_1} - first hypothesis tweet.
- $D_{T_{h_1}}$ - index for the medical condition in T_{h_1} . Indices used: 0 - Flu , 1 - Ebola , 2 - Measles , 3 - Diarrhea, 4 - Zika.
- $M_{T_{h_1}}$ - classification label of the tweet T_{h_1} : 0 - not related with a medical condition, 1 - related with a medical condition, 2 - ambiguous.
- T_{h_2} - second hypothesis tweet.
- $D_{T_{h_2}}$ - index for the medical condition in T_{h_2} . Indices used: 0 - Flu , 1 - Ebola , 2 - Measles , 3 - Diarrhea, 4 - Zika.
- $M_{T_{h_2}}$ - classification label of the tweet T_{h_2} : 0 - not related with a medical condition, 1 - related with a medical condition, 2 - ambiguous.
- $\text{rank}(T_p, T_{h_1})$ - similarity ranking between T_p and T_{h_1} as given by the labelers.
- $\text{rank}(T_p, T_{h_2})$ - similarity ranking between T_p and T_{h_2} as given by the labelers.
- L - classification label for relative comparison between tweets: 1 - if $\text{rank}(T_p, T_{h_1}) \geq \text{rank}(T_p, T_{h_2})$, 0 - if $\text{rank}(T_p, T_{h_1}) < \text{rank}(T_p, T_{h_2})$.

4.5 Architecture of the Learning Model

The tuples described in the previous section are used in our method, and we train a model M' on the following *classification task*: **determine which tweet is more similar to T_p : T_{h_1} or T_{h_2}** . In doing so, we will be training a sub-model M that does a *regression task*: compute the $rank(T_p, T_{h_i})$ between the premise and a hypothesis. **M will become the model that is actually used in production to rank tweets.** Our approach is a novel adaptation of a triplet-based network architecture [2], modified to classify if tweets are similar or not. It is based on the level of similarity between tweets and their relatedness with a disease. The input to train the algorithm is given in a Comma Separated Values (CSV) file that contains tweets and its label values (see section 5.5) in the tuple form presented in section 4.4. Input data passes through a process to finally obtain the measure of similarity between a pair of tweets.

Figure 4.2 depicts the organization of the DL model that we used. The model starts with an embedding layer. Each of the tweets in a tuple passes through an embedding process to convert sentences into a matrix of multidimensional vectors, each row representing words. This embedding layer is a modified siamese network since it is the same layer and weights are used to compute the embedding of the premise tweet T_p and hypothesis tweets T_{h_1} and T_{h_2} . Next, we have the Combiner Layer, where we use different approaches such RNN and CNN to extract features from the data. We use several combinations of hyperparameters until we get better results.

In the relevance layer we add a second input, called auxiliary input, that contains the information about the medical keyword of the tweet and the label indicating if the tweet is actually related with the medical condition or not. Again, we use the siamese network concept to pass this information for each tweet in the input. Using the auxiliary input and the output of the Combiner Layer, we calculate the relevance between the premise tweet and hypothesis tweet. This value becomes the rank of the premise and hypothesis.

Up to this point, the neural network has trained M , the model that computes the similarity ranking and the one we actually want in production. But the process is not finished. The final network component is the classification layer that is composed of a sequence of dense layers that end with a sigmoid classifier to finally get an output with value of zero or one. A label of 1 means that the first hypothesis T_{h_1} is more similar to the premise T_p , and 0 means that the second hypothesis T_{h_2} is more similar to the premise T_p . This neural network is model M' that we have referred as the “bigger” model to be trained. Notice how training M' indirectly trains model M , since we need to have the

similarity ranks in order to make the comparison between T_p, T_{h_1} and T_{h_2} , and emit the final label of the classification task.

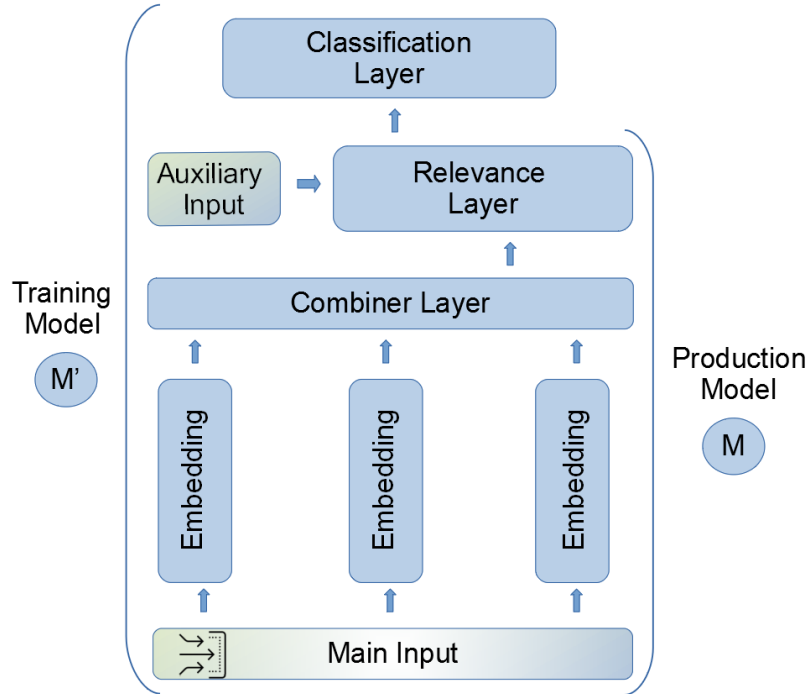


Fig 4.2: Model Architecture

4.5.1 Data Setup

The data obtained after the labeling step is prepared to be compatible with the neural network inputs and output labels. The inputs to the neural network are the tweet, the disease type, and disease-related labeling; these inputs are required for each member of triplet (T_p, T_{h_1}, T_{h_2}) . Additionally, we need the labels for the relevance between premise and the two hypotheses. These are the rankings that the labelers completed, and become the secondary output of the network. The main output of the network is the final binary classification of similarity which indicates which hypothesis tweet is more similar to the premise tweet. The total of data examples available was 11,425 tweets.

Each tweet is converted into a matrix, where a row is a vector in an n-dimensional embedding space that represents a word. The tweets form the main input to the network. The disease group label and the disease-relatedness labels form the auxiliary input. Both of these features were binarized to change from a categorical feature. In the next tweet, we illustrate how we created the representation of the auxiliary input.

Example 4 *“It is march with streets are fully covered in snow I am drinking hot milk with cocoa butter and recovering from a flu I just turned my Christmas tree lights on yes, it is still here”*

In the example above, the tweet mentions “flu”, which has index 0. Suppose we have the next conditions: 0 - Flu, 1- Measles, 2- Diarrhea, and 3 - Ebola. We need a bit vector with 4 entries for this. Flu is then represented as $[1, 0, 0, 0]$. This is shown on the upper left part of Figure 4.3. We have three labels for the relatedness to medical condition: 0 - not related, 1 - related, and 2- ambiguous. Here, we need a bit vector with 3 entries. Since the tweet is related with a medical condition, which is label 1, its representation would be $[0, 1, 0]$. This is shown on the upper right part of Figure 4.3. Then, for this tweet, its auxiliary representation would be the concatenation of these two bit vectors: $[1, 0, 0, 0, 0, 1, 0]$. This process is done for each of the three tweets (T_p, T_{h_1}, T_{h_2}) .

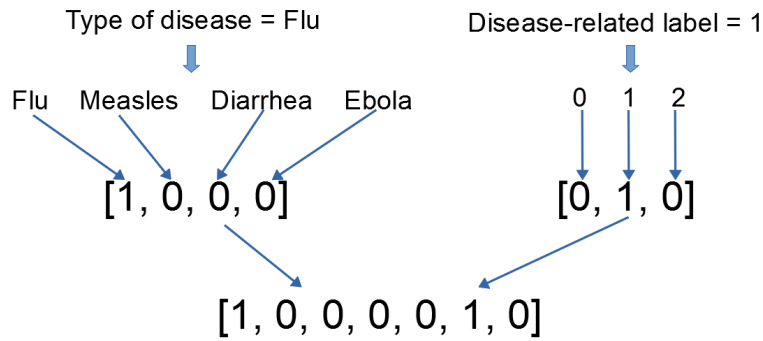


Fig 4.3: Auxiliary Input Representation

The main input of the neural network are the three tweets converted into their embedding matrices. For this purpose, our embedding layer uses GloVe embedding [18], an unsupervised algorithm to get a vector representation for each word in the tweets. We used a pre-trained embedding that was trained with 2 billion of tweets. It is formed by a vocabulary of 1.2 million terms with vectors of 25, 50, 100 and 200 dimensions. For our models, we used the versions with 50 and 200 dimensions. Before feeding them to the neural network, the tweets were passed by a preprocessing step to remove stop words and lemmatized each word to the root lemma.

4.5.2 Combiner Layer

The purpose of the combiner layer is to enable the neural network to build useful features to help it figure out how to find the similarity between tweets. In the early days of

neural networks, features had to be engineered by hand and passed as inputs to the neural network. In contrast, modern DL approaches rely on adding enough layer to the network so it can create the features itself. The input to the combiner network are the embedding of the tweets. In our approach, we implement three different combinations of algorithms that include RNN and CNN. Each method was implemented with variation in the hyperparameters depending of each algorithm. Finding the best hyperparameter combinations is not a trivial task. We based our experiments on the result of related researchers [29], and then we created our own combination, is shown in Table 4.1.

The hyperparameters variation was in epochs, learning rate, batch size, and optimizer. The hyperparameters mentioned above are not learned by the network, those ones have predefined values set before training process.

Table 4.1 Hyperparameter Combinations

Hyperparameters	Values
Learning rate	0.001
Epochs	10, 20
Batch size	32, 128
Optimizer	RMSprop, Adam

4.5.2.1 RNN Combiner

Using recurrent networks, we have two different combinations. The first implementation was a combination of two LSTM networks; they are networks with loops in them, allowing information to persist. The first LSTM network contains 128 neuron units and returns all sequences by each time step. The next LSTM network also has 128 units, but only return the final sequence (accumulated output). The structure of this network is shown in Figure 4.4. The same stack of LSTM networks is used to process the three embeddings of the tweets. E_1 is the embedding of T_p , E_2 is the embedding of T_{h_1} , and E_3 is the embedding of T_{h_2} .

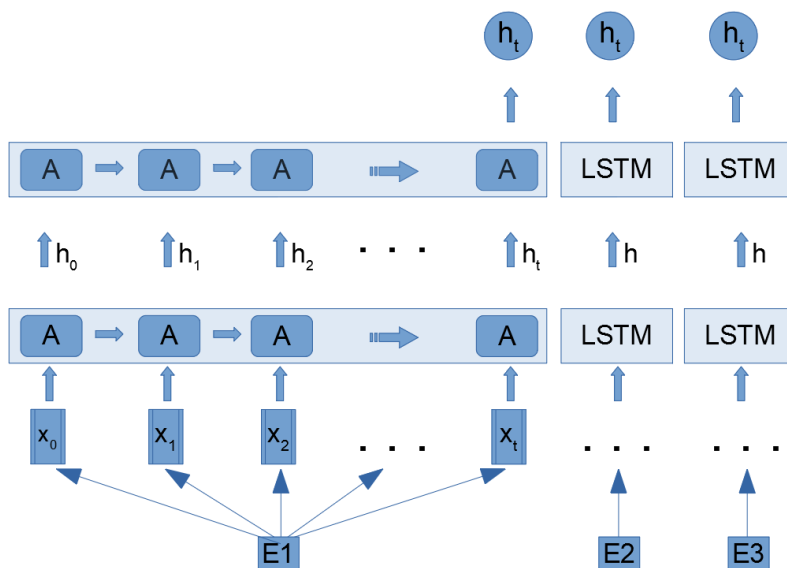


Fig 4.4: Regular LSTM Network

Figure 4.4 shows a double LSTM network, where embedded tweets are the inputs and the output is a tensor with the accumulate sequence. This process is the same for each embedding vector.

Our second network implementation is a variant of LSTM networks. In this implementation we use two sequential bidirectional LSTM networks, that consist of two independent LSTM networks. The architecture is showed in Figure 4.5. In our bidirectional LSTM, one LSTM is used to process the tweets in regular word order, and the second one in reverse word order. Bidirectional LSTM are used to make predictions with both past and future context.

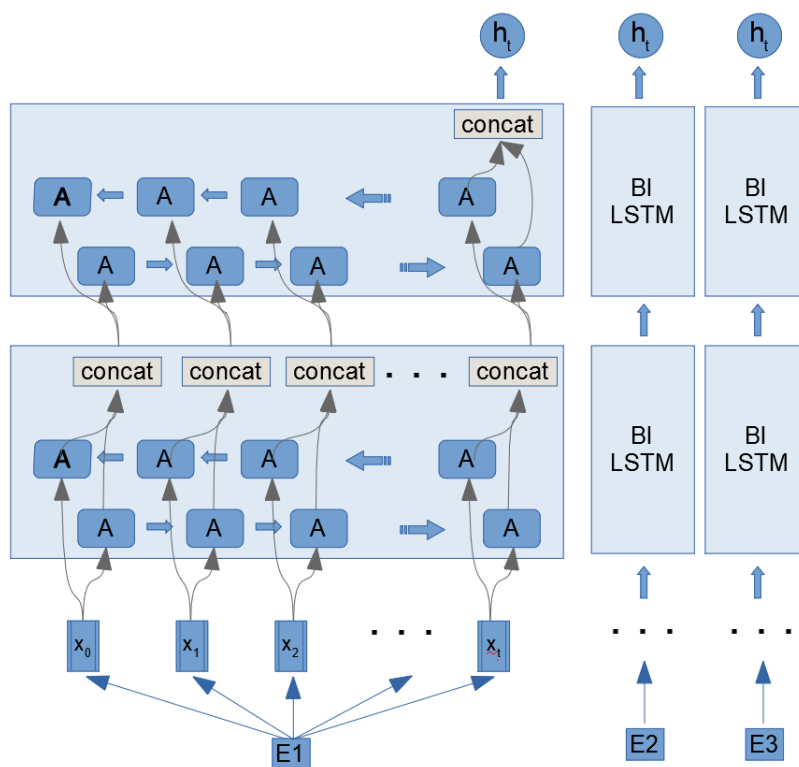


Fig 4.5: BiLSTM Network

4.5.2.2 CNN Combiner

The role of a CNN is reduce an initial sample given as a matrix into a form which is easier to process without losing features. The basic operation in this network is called “convolution”. This operation has the objective to extract the high level features applying a filter to an input. The repeated application of the same filter results in a map of activations indicating the locations and strength of a detected feature. In this method we use the concept of inception networks. The key idea in an inception module is to deploy convolutions with multiple filters, and pooling operations in parallel within the same layer [36]. A basic inception module is showed in Figure 4.6

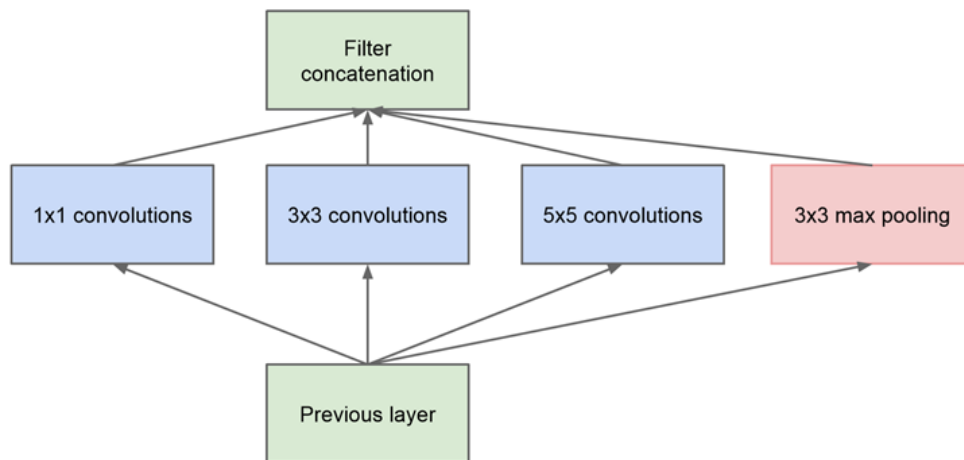


Fig 4.6: Inception Module as depicted in [36].

Our implementation of CNN using inception layers is shown in Figure 4.7. This network follows the basic concept of an inception module using filters of 1x1, 3x3, and 5x5 convolutions. Filters with length of 5x5 passed through a zero padding layer used such that the output has the same length as the original input. Next the results of each filter are concatenated and finally we obtain a output containing the high level features of each tweet.

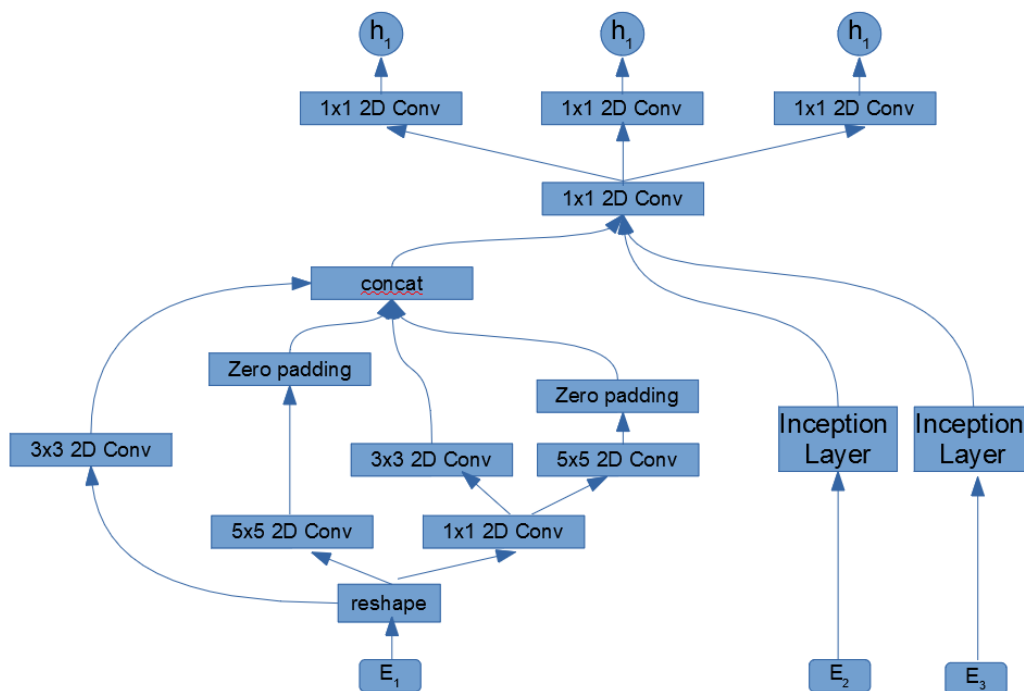


Fig 4.7: Inception CNN

4.5.3 Relevance Layer

The input for the relevance layer are the three outputs of the combiner layer: C_1, C_2 and C_3 . These tensors represent the output of the combiner layer for E_1, E_2 and E_3 , respectively. In the relevance layer the network tries to learn the relevance rank between tweets: $rank(T_p, T_{h_1})$ and $rank(T_p, T_{h_2})$. The full structure of the relevance component is shown in Figure 4.8.

To extract relationships between tweets, various methods concatenations are applied :

- $D_1 = \text{concat}(C_1 * C_2, |C_1 - C_2|, C_1, C_2)$ - combines features from T_p and T_{h_1}
- $D_2 = \text{concat}(C_1 * C_3, |C_1 - C_3|, C_1, C_3)$ - combines features from T_p and T_{h_2}

Where $\text{concat}()$ is a concatenation operation. D_1 is a feature tensor build by concatenating: a) element-wise product of C_1 and C_2 , b) absolute difference of C_1 and C_2 , c) C_1 itself, and d) C_2 itself. This idea is adapted from the work in [20]. The same process is done with D_2 . Each tensor D_i is then concatenated with the auxiliary inputs the pair of tweets used to form D_i . This resulting tensor is passed through a dense layer of 128 units with $ReLU$ activation and then goes into a dense layer of one unit with no activation. The result represents the similarity r_i between the premise tweets and the hypothesis tweet T_{h_i} . The value r_1 gives the similarity between T_p and T_{h_1} , while r_2 gives the similarity between T_p and T_{h_2} . These two values are secondary outputs to the model, and are also used in the final classification layer.

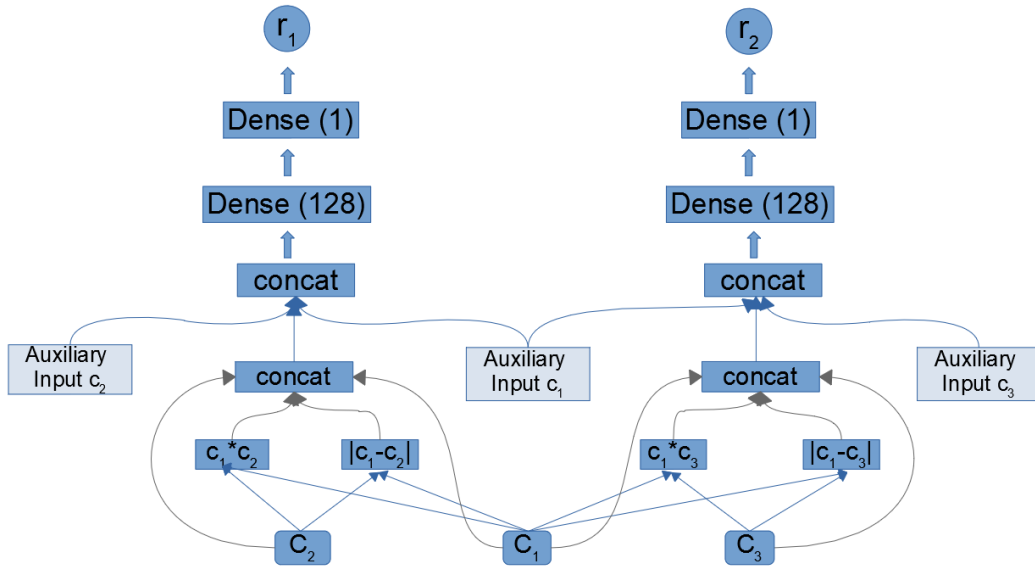


Fig 4.8: Relevance Layer

4.5.4 Classification Layer

The classification layer takes as input the relevance layer outputs and concatenates them. Next, this tensor is fed into a sequence of three dense layers of 16, 8, and 1 units respectively. The dense layers with 16 and 8 units have a ReLU activations. In the final dense layer, we use a sigmoid activation to obtain a binary output. The final output can take two values: a) 1 means the first hypothesis T_{h_1} is more similar to the premise T_p , and b) 0 when second hypothesis T_{h_2} is more similar to premise T_p . Figure 4.9 shows the architecture of this component.

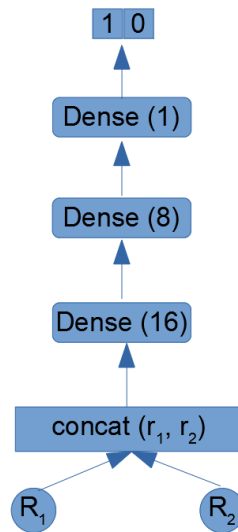


Fig 4.9: Classification Layer

4.6 Metrics

Our model has one main output and two secondary outputs:

- **Main output** - classification label to indicate which hypothesis tweet is more similar to the premise.
- **Secondary output #1** - similarity between T_p and T_{h_1} : $rank(T_p, T_{h_1})$.
- **Secondary output #2** - similarity between T_p and T_{h_2} : $rank(T_p, T_{h_2})$.

We use *accuracy* to measure the quality of our models on the main output since this is a classification task. Likewise, we use *Mean Squared Error* to measure the quality of our models on the secondary outputs since these are regression tasks.

Chapter 5

Performance Evaluation

In this chapter we evaluate the quality of our DL models by measuring their accuracy and mean squared errors on our data sets.

5.1 Hardware

In this research we use two different environments to run our experiments. The cluster distribution presented in Figure 5.1 shows the components of the THS project that was configured as described in Section 4.1. The master-node and edge-node were set with Ubuntu 16.04 LTS Operating System (OS) and data-nodes with Ubuntu 14.04 LTS OS. These nodes are installed and run on bare metal. All operations are executed in the edge-node that connect the users with Big Data environment.

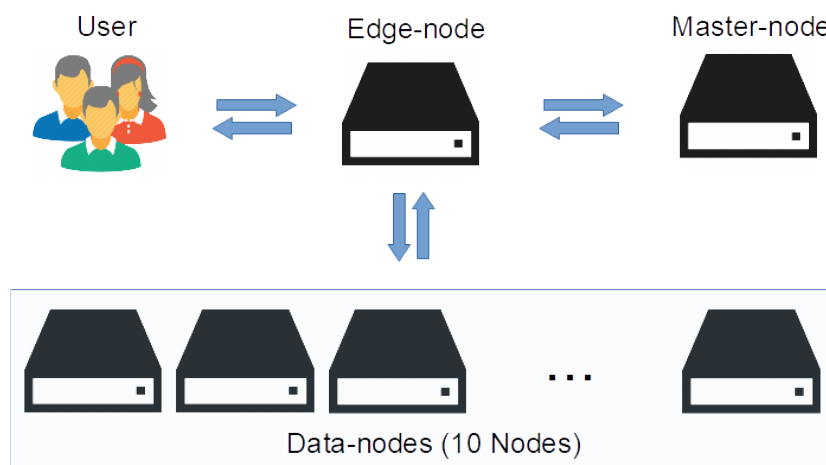


Fig 5.1: THS Cluster

The hardware specification by each node is shown in table 5.1. The total cluster storage is 2.9 TB and 80 GB of memory to processing data (excluding master-node and edge-node). **This environment was used for tweet capture, filtering, and storage**

Table 5.1 THS Cluster Node

Hardware	Description
Hard disk	297 GB
RAM	8 GB
Processor	Intel(R) Xeon(R) E3120 @ 3.16GHz
GPU	None

The other environment with important resources to run our experiments was Chameleon Cloud, an online large-scale platform for research, with a variety of resources in Hardware and Software. For this project we **used Chameleon nodes with Graphic Processing Unit (GPU) resources to train our DL models**. These nodes ran in Ubuntu 16.04 LTS on bare metal. The OS image used for the project is a custom image created by THS project. The hardware specification is described next in table 5.2.

Table 5.2 Chameleon Cloud Custom Node

Hardware	Description
Hard disk	207 GB
RAM	128 GB
Processor	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30 GHz x 2
GPU	Tesla P100 - 32GB

5.2 Software

The software resources used for this research are the following:

- **Python:** A programming language used in research fields, for its simplicity and power. It is compatible with tools used in this research: TensorFlow and Keras. All the code in this project is written in Python language.
- **Twitter API:** A tool to get data in real time from twitter for our research.
- **HDFS:** This software allows a distributed processing of massive amounts of data, running in commodity hardware [32].

- **HIVE:** A data warehouse software for large datasets in a distributed storage. For querying, it use use Structured Query Language (SQL) syntax [35].
- **KAFKA:** It is a streaming platform to store streams of records and enables to process them asynchronously. It can be implemented in standalone mode or cluster mode. Each stream store is called a “topic” [33].
- **SPARK:** Apache Spark is a fast, general purpose and unified analytic engine for large-scale data processing. It is compatible with JAVA, Scala, Python, and R. Spark support structured data processing, ML, graph processing, and stream processing of live data [34].
- **Tensorflow:** Google’s Open source ML library for building and training complex neural network models [31].
- **Keras:** A neural network API, that allows easy and fast experimentation, and supports CNN and RNN. This tool works with Python and can be used on Central Processing Unit (CPU) and GPU [30]. **All ML code in this project was implemented with Keras.**
- **Compute Unified Device Architecture (CUDA):** A programming model developed by NVIDIA; it is a parallel computing programming on GPU, the sequential computes on thousands of cores in parallel mode, optimizing the performance of tasks [37]. TensorFlow uses CUDA to implement its neural networks.
- **Torch:** A framework to support ML algorithms using GPU in a efficient way. This tool seeks to facilitate and speed up the implementation of neural network libraries and optimization packages.

The versions of the tools implemented in THS project are showed in Table 5.3 whereas the versions of software packages installed in both environments are presented in table 5.4.

Table 5.3 Big Data Tools in THS System

Software	Version
Hadoop & Yarn	2.7
Hive	2.2
Spark	2.1
Kafka	0.10.1

Table 5.4 Version of Software in THS Nodes

Software	Version
TensorFlow	1.12.0
TensorBoard	1.12.2
Keras	2.2.2
Scikit-learn	0.19.1
Natural Language Toolkit (NLTK)	3.4
SciPy	1.1.0
Torch	1.0.1
CUDA	9.0

5.3 Data Collection for Training Models

The process to collect data and prepare it for processing as input to our models is shown in Figure 5.2 and it is described in the steps following.

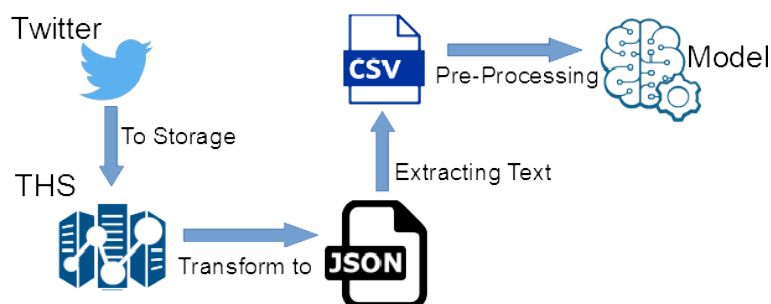


Fig 5.2: Data Collection Process

- **Step 1:** To get data from Twitter we used a Twitter Streaming API, and we collected data and processed it through THS system where the tweets are stored in a Hive Database.
- **Step 2:** Running a python script we got data from the “raw_tweet” table of the THS system. Here, we filtered data by specific diseases and by date, using Spark tools. The tweets are saved in a JavaScript Object Notation (JSON) file because originally tweets have a JSON format. This facilitates the next step of processing.
- **Step 3:** Using the JSON file created in the previous step, we filtered data and only took the tweet text (sentence itself) because our project only needs to work with the text written by the user. Also, we threw away repeated tweets. This filtered data was saved in a CSV file to next processing step.

- **Step 4:** In this step the data passes to the pre-processing step to clean and filter data before deliver data to labeling step. This process is described in Section 5.4
- **Step 5:** The data collected is ready for labeling, and then for input to our model.

5.4 Data Pre-processing

Data collected from social networks contains misspelling, slang, or no textual information, that makes it less structured and informal. This is the reason for cleaning the data as a previous step to build our model. This stage is composed by three demarcated steps: Pre-processing before labeling, labeling task, and final pre-processing. These steps are shown in Figure 5.3.

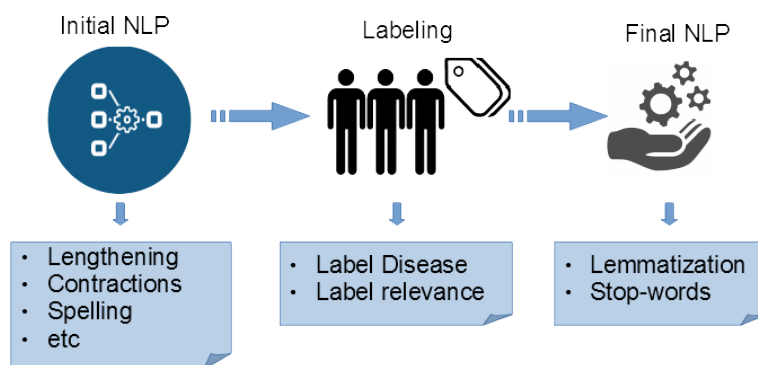


Fig 5.3: Data Pre-processing

In this section we will describe the first and last step of processing. The labeling step will be described in section 5.5. Before using tweets directly or labeling, it is necessary to filter and clean the data [9, 38, 39]. The operations to prepare data are described below.

- **Remove line breaks:** It is necessary to clean line breaks because they can be generating more than one row from the same tweet and we can lose context and meaning.
- **Remove links, hashtags and mentions:** Social media is informal and contains no textual information; this is why we cleaned no-relevant data such as links, hashtags, and mentions [38].
- **Character unescape:** We replaced source code based on hexadecimal code with the character that it represents to be understandable and compatible with human language.
- **Reduce lengthening:** Many typo errors are presented in the text like repetition of a character, many times to express emphasis or simply because there exists a

typographic mistake. For example, the word “fluuu” represents the next word “flu”. In cases like this one, it needs to be fixed, and replaced with the right word.

- **Expand contractions:** To ensure a correct representation of each word, we expanded all contraction to their original form.
- **Keep text and Number:** After the previous operation, we removed all characters which are noisy to the ML algorithm. We just conserved text and numbers.
- **Spelling corrections:** It was necessary a spelling corrector to fix typographical errors and help to avoid misunderstanding in labeling step or to process it as an unknown word in the vectorization step.

Cleaned data obtained from the previous process is ready to deliver to labeler users, from which main and auxiliary label inputs are built. This step is described in section 5.5.

The final NLP steps are:

- **Lemmatization:** It is a process to find a lemma for each word in a tweet. This refers to converting all inflected forms in its base form. For example, the words: “taking”, and “takes” refers to base form of the word “take”. Lemmatization considers the context and meaning to convert a word to its base form. For example in the word “caring” a simple stemming process will cutoff termination “ing” and return “car”; in change a lemmatization process will return the word “care”. The better known tools are NLTK, textBlob, spaCy, pattern, and Stanford CoreNLP.
- **Remove stop-words:** The final operation of the processing step is removing the stop-words (e.g. articles, and pronouns) and just keep words with high meaning.

5.5 Data Labeling

Since our model follows a supervised approach, it is necessary to use labeled data. In this project we have two different kinds of labeling. The first one is focused in classifying the tweets if they are related or no to a disease. The other type refers to set a measure of similarity between tweets. We now describe how they were built.

5.5.1 Disease-Related Labeling

The training set is composed of 11,937 tweets and it was labeled by five members of the THS team. Labels depends on the classification in term of similar meaning and their relatedness with specific topics about health diseases. There are three possible label values

for each tweet.

- **0:** This label represents a tweet that is not related to a disease.
- **1:** It is a tweet with meaning related to a disease.
- **2:** Tweets have an ambiguous meaning, difficult to classify.

5.5.2 Labeling for Level of Relevance (rank)

To label for relevance of similarity between tweets we built triplets (collection of three sets). The first dataset consisted of 11,425 triplets (T_p, T_{h_1}, T_{h_2}). All the tweets in these triplets were known to be related with a medical conditions and had one of the four target keywords: Ebola, Flu, Measles, or Diarrhea. Labelers were asked to rank the similarity between T_p and T_{h_1} and between T_p and T_{h_2} . There were 54 labelers, which were students from UPRM. Triplets were equally divided between them.

Labelers were instructed to give a score between 1 and 4. The rules that were considered to manually labeling the relevance between tweets were the following:

1. Score 4 - When tweets discuss the same disease:

- **Tweet 1:** *There is a measles outbreak in japan.*
- **Tweet 2:** *Japanese fear measles outbreak in progress.*

2. Score 3 - Tweets discuss the same disease, but the events are not related:

- **Tweet 1:** *Government intensifies educational campaign against Ebola.*
- **Tweet 2:** *Many countries launch initiatives to raise awareness about Ebola.*

3. Score 2 - Tweets discuss the same disease at the same location:

- **Tweet 1:** *Outbreak report of Zika at Ponce.*
- **Tweet 2:** *Government of Ponce warns about outbreak of Zika.*

Labelers were given two options for giving a score of one (1); if the tweets

4. Score 1 - Described different symptoms of the same disease:

- **Tweet 1:** *Big headache and fever due do you the flu.*
- **Tweet 2:** *I have been two days in bed feeling ill, due the flu, or,*

5. Score 1 - If the tweets referred to similar action in topics related to diseases:

- **Tweet 1:** *Government recommends getting the flu vaccine.*
- **Tweet 2:** *Today I went to the pharmacy to get the flu vaccine.*

We augmented the data by adding two additional group of triplets. In the first group, one of the hypothesis tweets, T_{h_i} has the same disease keyword as T_p but is not related to a medical condition (class 0). In this case, the ranking of this tweet with respect to T_p is set to 0. In the second group, one of the hypothesis tweets, T_{h_i} has a different disease keyword as that T_p . In this case, the ranking of this tweet with respect to T_p is also set to 0. In total, we grew the training data set to 11,425 tweets. Based on the ranking of the tweets we provide a final label L . This label is a classification label for relative comparison between tweets: 1 - if $rank(T_p, T_{h_1}) \geq rank(T_p, T_{h_2})$, 0 - if $rank(T_p, T_{h_1}) < rank(T_p, T_{h_2})$.

5.6 Experiment Setup

Experiments were set up in two environments with different resources: Chameleon Cloud and THS cluster. Data acquisition and pre-processing was done with the THS cluster. Model training was done in Chameleon. Figure 5.4 shows an overview of the setup structure of the whole project.

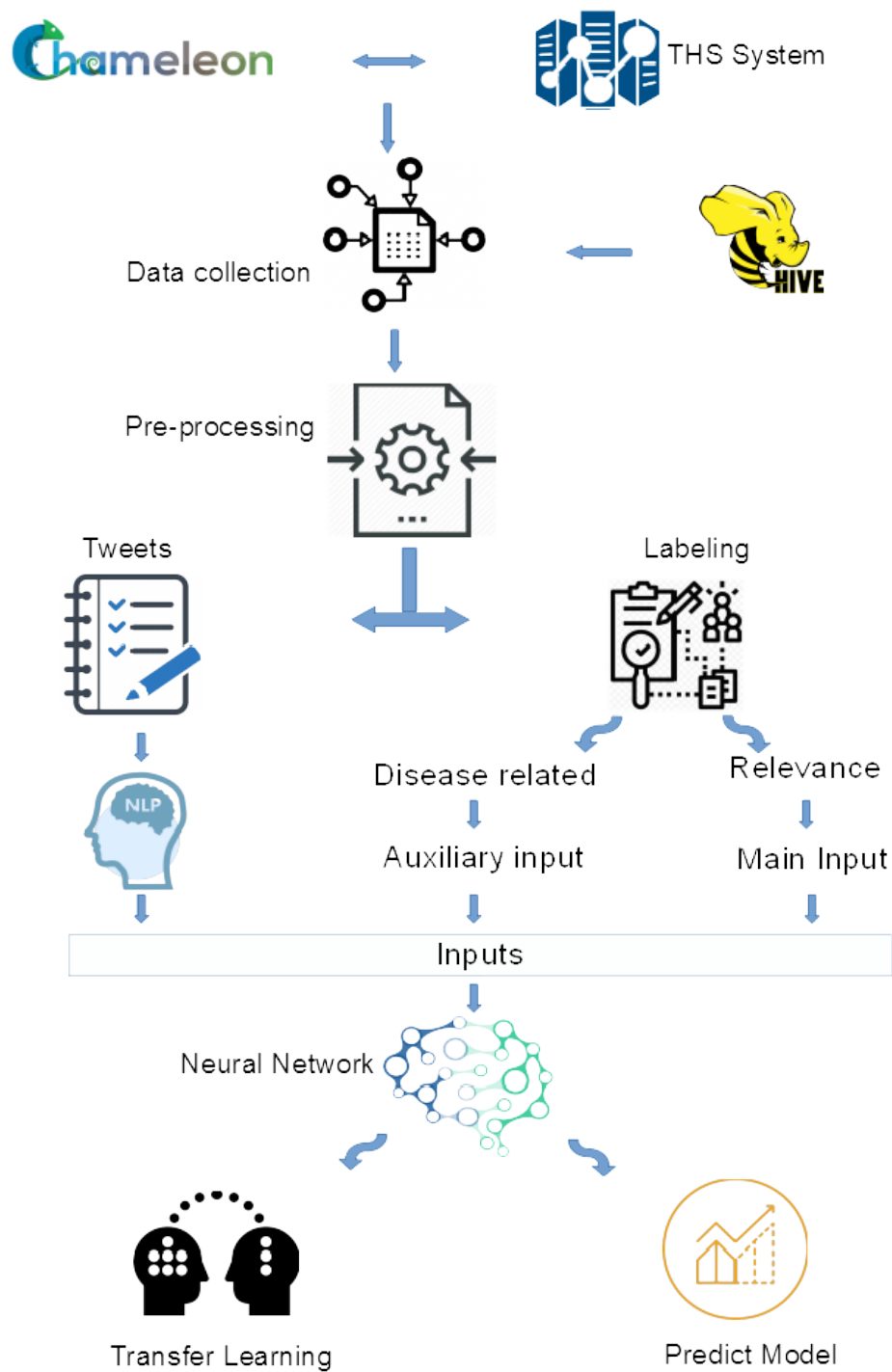


Fig 5.4: Model Setup Overview

5.6.1 Experimental Methods

To train our algorithm, we mix the rows randomly to prevent order dependencies between runs and assign each example to one of these sub-sets:

- **Training set:** This group comprehends 80% of the initial data, that is 9,140 triplets. This data is used to train all combinations of algorithms presented.
- **Validation set:** The remaining data is the 20% that comprehends 2,285 samples. This data is used to validate the model by fitting the weights of parameters to find the best performance.

Usually models of ML assume that input data has an uniform distribution between classes and do not consider the presence of imbalance data. In our case, our data is imbalanced because there are more examples classified with label 0. In our case we need to adjust our model to work with imbalanced data, because the model could be predicting in favor of the majority class. The approach taken to avoid this issue was to use a built-in functionality in Skitlearn library. This tool returns us the weights penalties for misclassifying in imbalanced classes. This approach provides a natural and straightforward method to set weight penalization.

Table 5.5 shows the penalties by each class. A higher value is given for the minority class and a lower value to the majority class.

Table 5.5 Triplets Distribution

Class	Penalty
1	1.181
0	0.867

5.6.2 Results for Models using RNN at the Combiner Layer

We divide RNN results in two parts. First, we show those related with LSTM combiner and the second with bidirectional RNN combiner. For RNN implementation we used an embedding vector of 50 and 200 dimensions. We used a larger embedding matrix of 200 dimensions to find out relations between tweets given to model more features. These embeddings were mentioned in section 4.5.1 are a pre-trained vectors of GloVe embedding.

5.6.2.1 Regular LSTM

In Table 5.6 we show the accuracy results for the LSTM combiner implementation. The tweet data is presented in two ways: an embedding vector taken without a preliminary

preprocessing and vectors built after removing stop-words and the lemmatization process.

Table 5.6 Accuracy for Regular LSTM

Embedding	Data Type	Epochs	Accuracy Training	Accuracy Validation
50 dimensions	All Text	10	0.89	0.89
50 dimensions	All Text	20	0.89	0.88
50 dimensions	No Stop words and lemmatized	10	0.89	0.90
50 dimensions	No Stop words and lemmatized	20	0.90	0.88
200 dimensions	All Text	10	0.89	0.89
200 dimensions	All Text	20	0.93	0.85
200 dimensions	No Stop words and lemmatized	10	0.90	0.88
200 dimensions	No Stop words and lemmatized	20	0.97	0.86

The results for accuracy metric are similar. The accuracy in the training step is in the range between 89% and 97%. The accuracy in the validation step is in the range between 85% and 90%. The next configuration has the better accuracy metrics:

- 50 dimensions, 10 epochs, after lemmatization step, and rmsprop optimizer with 89% and 90% of accuracy in training and validation, respectively.
- 200 dimensions, 20 epochs, after lemmatization process, rmsprop optimizer with 97% and 86% of accuracy for training and validation, respectively.

The case of 50 dimensions, 10 epochs, rmsprop optimizer, and after lemmatization step provided best accuracy on validation and it is the selected model.

In table 5.7 we show measures for the loss functions. We used Mean Squared Error (MSE) loss function in the ranking outputs and cross-entropy loss function for final

classification of triplets. MSE measures the average of the squares of the errors in our model. For the best model the MSE metrics were 0.37 and 0.34 of training and validation, respectively.

Table 5.7 Loss Metrics for Regular LSTM

Dims	Data Type	Epochs	MSE Training	Cross-Entropy Training	MSE Val- idation	Cross-Entropy Training
50	All Text	10	0.37	0.23	0.34	0.23
50	All Text	20	0.36	0.21	0.36	0.24
50	No Stop words and lemmatized	10	0.37	0.23	0.34	0.22
50	No Stop words and lemmatized	20	0.36	0.21	0.35	0.24
200	All Text	10	0.38	0.22	0.36	0.24
200	All Text	20	0.38	0.15	0.38	0.35
200	No Stop words and lemmatized	10	0.38	0.20	0.36	0.25
200	No Stop words and lemmatized	20	0.41	0.07	0.41	0.43

5.6.2.2 Bidirectional RNN

In table 5.8 is presented the accuracy outputs for Bidirectional RNN implementation. As we explained in Section 4.5.2, our Bidirectional network is composed by two bidirectional LSTM networks.

Table 5.8 Accuracy Bidirectional Network

Embedding	Data Type	Epochs	Accuracy Training	Accuracy Validation
50 dimensions	All Text	10	0.96	0.86
50 dimensions	All Text	20	0.99	0.86
50 dimensions	No Stop words and lemmatized	10	0.97	0.86
50 dimensions	No Stop words and lemmatized	20	0.99	0.85
200 dimensions	All Text	10	0.99	0.86
200 dimensions	All Text	20	0.99	0.86
200 dimensions	No Stop words and lemmatized	10	0.99	0.86
200 dimensions	No Stop words and lemmatized	20	0.99	0.87

The results for accuracy metric are similar. The accuracy in training step is in the range between 96% and 99%. The accuracy in validation step is in the range between 85% and 87%. The better accuracy metrics have the next configuration:

- 200 dimensions, 20 epochs and after lemmatization step, rmsprop optimizer with an accuracy of 99% and 87% in training and validation, respectively.

Notice that the models with bidirectional LSTM in the combiner tend to overfit the input.

In table 5.9 we show measures for loss functions. We used Mean Squared Error (MSE) loss function in the relevance rank outputs and cross-entropy loss function for final classification of triplets.

Table 5.9 Loss Bidirectional Network

Dims	Data Type	Epochs	MSE Training	Cross-Entropy Training	MSE Val- idation	Cross-Entropy Training
50	All Text	10	0.39	0.10	0.40	0.41
50	All Text	20	0.35	0.02	0.41	0.77
50	No Stop words and lemmatized	10	0.39	0.08	0.40	0.44
50	No Stop words and lemmatized	20	0.35	0.02	0.39	0.80
200	All Text	10	0.38	0.03	0.43	0.61
200	All Text	20	0.31	0.01	0.36	0.76
200	No Stop words and lemmatized	10	0.36	0.02	0.39	0.66
200	No Stop words and lemmatized	20	0.28	0.01	0.35	0.78

5.6.3 Results for Models using CNN at the Combiner Layer

For the CNN implementation we also used an embedding vector of 50 and 200 dimensions. We used the Inception network approach, a complex and heavily engineered CNN network. It used a lot of tricks to push performance, in terms of speed and accuracy. The network essentially would get wider rather than deeper, in order to tackle the issue of overfitting. The network structure is described in section 4.5.2.

In table 5.10 is presented the accuracy outputs for Inception CNN implementation.

Table 5.10 Accuracy CNN Model

Embedding	Data Type	Epochs	Accuracy Training	Accuracy Validation
50 dimensions	All Text	10	0.97	0.85
50 dimensions	All Text	20	0.99	0.86
50 dimensions	No Stop words and lemmatized	10	0.95	0.85
50 dimensions	No Stop words and lemmatized	20	0.99	0.84
200 dimensions	All Text	10	0.98	0.84
200 dimensions	All Text	20	0.99	0.87
200 dimensions	No Stop words and lemmatized	10	0.97	0.86
200 dimensions	No Stop words and lemmatized	20	0.99	0.86

In general the metrics for accuracy are very close. The higher value was when the tweets only passed the first pre-processing (without remove stop-words or lemmatization process). The accuracy in training step is in the range between 95% and 99%. The accuracy in validation step is in the range between 84% and 87%. The better accuracy metrics have the next configuration:

- 50 dimensions, 20 epochs and before lemmatization step, rmsprop optimizer with an accuracy of 99% and 86% in training and validation, respectively.
- 200 dimensions, 20 epochs and before lemmatization step, adam optimizer with an accuracy of 99% and 87% in training and validation, respectively.

In table 5.11 we show measures for loss functions. We used Mean Squared Error (MSE) loss function in relevance parameters and cross-entropy loss function for final classification of triplets.

Table 5.11 Loss CNN Model

Dims	Data Type	Epochs	MSE Training	Cross-Entropy Training	MSE Val-idation	Cross-Entropy Training
50	All Text	10	0.42	0.08	0.47	0.43
50	All Text	20	0.24	0.02	0.34	0.75
50	No Stop words and lemmatized	10	1.48	0.12	1.41	0.37
50	No Stop words and lemmatized	20	1.34	0.03	1.29	0.66
200	All Text	10	1.56	0.06	1.54	0.53
200	All Text	20	1.33	0.02	1.35	0.66
200	No Stop words and lemmatized	10	1.51	0.08	1.48	0.45
200	No Stop words and lemmatized	20	1.31	0.03	1.29	0.59

5.6.4 Discussion of Results

In the results for models the RNN combiner we concluded that the best performing in accuracy is a model regular LSTM combiner with an embedding using 50 dimensions, with text processed after lemmatization, 10 epochs, batch size of 32 and rmsprop optimizer. The results show a training accuracy of 89% in training and 90% in validation step. The loss metrics were 0.37 and 0.34 for MSE in training and validation. The cross-entropy loss function in this configuration had 0.23 for training step and 0.22 in validation.

In the case of CNN approach the best model are close among them and not distant from the RNN results, both having a high accuracy for training. The configuration of hyperparameters for the best model had an embedding with 200 dimensions, text before lemmatization and without remove stop words, batch size of 32 and adam optimizer. The measured of accuracy in training was 99% and 87% for validation. MSE in training had a value of 1.33 and 1.3. On the other hand Cross-entropy loss had 0.02 in training and 0.66

in validation.

Comparing RNN and CNN results, we can see that RNN with two LSTM network joined sequentially have more consistency in training and validation accuracy, because the metric for accuracy are 89% and 90% in training and validation, respectively. Notice that the best result of the bidirectional RNN approach have the same measures for accuracy as the best models with CNN. The tradeoff is that CNN required less time to train. In general terms the order of training time of the three approaches presented is the next, inception CNN is the fastest, next is the regular LSTM in second place, and finally, with worst time execution, is Bidirectional LSTM networks.

Chapter 6

Conclusion and Future Work

In this project we used social networks as an application to get valuable information about health topics. In our case we used Twitter and its vast amounts of data on different topics. We illustrated how to use tweet data about medical conditions to build models that are able to compute the similarity in tweets. We use the THS system to store data and then use it as our input data.

We used neural network employing supervised learning approach in text similarity. We showed that trained models with labeled data in sentence similarity have good performance to be widely adopted for tweet similarity and in others NLP tasks. We build models with deep CNN and RNN obtaining good results with approximately 90% on validation step.

We trained a model to compute the similarity rank between two tweets, by first training a bigger model M' capable of classifying relative similarity in triplets of tweets. We trained a bigger model that classifies which of two tweets is more similar to a premise tweet, but we use a sub-task (sub-model) M to predict a relevance metric between two tweets. This model M can then be used to produce similarity scores and order a list of tweets by similarity respect to a premise tweet.

In order to validate our model we presented a performance study on the DL similarity models with a data sets consisting of 11,425 examples, and we used that data set to train our models. Our results showed that we can achieve 90% accuracy on the task of classifying which of two tweets is more similar to a premise tweet. The sub-model M trained have a 0.34 units of Mean Squared Error on a decreasing loss.

Future work will be focused in training models with more data to avoid problems of overfitting, which enable us compare with the previous models. We plan on recruiting a team of experts with knowledge in Public Health to create a labelled data set for disease relevance and another to scoring the similarity between tweets. We plan on building

this dataset with equally distributed labels for the relevance metric. Another idea is to explore new architecture models in RNN and CNN approaches to using in combiner layer and determine if they improve in accuracy performance. Also, there is the idea of using distance metrics, like the Frobenius distance or cosine similarity in the relevance layer to have to different ways to measure the similarity.

Bibliography

- [1] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2010.
- [2] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. *CoRR*, abs/1404.4661, 2014.
- [3] Aoife D’Arcy John D. Kelleher, Brian Mac Namee. *Fundamentals of Machine Learning for Predictive Data Analytics*. The MIT Press, 2015.
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [5] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [6] Nils J. Nilson. *Introduction to Machine Learning*. Stanford University, 1998.
- [7] Andrew Ng. *Machine learning*, 2019.
- [8] Kimberly Nevala. *The Machine Learning Primer*. SAS Institute Inc., 2016.
- [9] A. S. Halibas, A. S. Shaffi, and M. A. K. V. Mohamed. Application of text classification and clustering of twitter data for business analytics. In *2018 Majan International Conference (MIC)*, pages 1–7, March 2018.
- [10] Kevin Gurney. *An Introduction to Neural Networks*. Taylor and Francis e-Library, 2004.
- [11] David Kriesel. *A Brief Introduction to Neural Networks*. dkriesel, 2005.

- [12] B. Chandra and R. K. Sharma. On improving recurrent neural network for image classification. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1904–1907, May 2017.
- [13] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.
- [14] Alex Graves. Supervised sequence labelling with recurrent neural networks, 2010.
- [15] Dan Jurafsky and James H. Martin. *Speech and language processing*, 2018.
- [16] Nitin Indukhya Fred J. Damerau. *Handbook of Natural Language Processing*. CRC Press, 2010.
- [17] Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 10 2013.
- [18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [19] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *CoRR*, abs/1506.06726, 2015.
- [20] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017.
- [21] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.
- [22] Kavita Ganesan. What is text similarity?, 11 2015.
- [23] Nitesh Pradhan, Manasi Gyanchandani, and Rajesh Wadhvani. A review on text similarity technique used in ir and its application. *International Journal of Computer Applications*, 120:29–34, 06 2015.

- [24] Goutam Majumder, Dr. Partha Pakray, Alexander Gelbukh, and David Pinto. Semantic textual similarity methods, tools, and applications: A survey. *Computacion y Sistemas*, 20:647–665, 12 2016.
- [25] Wael Gomaa and Aly Fahmy. A survey of text similarity approaches. *international journal of Computer Applications*, 68, 04 2013.
- [26] S. Zhang, X. Zheng, and C. Hu. A survey of semantic similarity and its application to social network analysis. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2362–2367, Oct 2015.
- [27] Tian Tian Zhu and Man Lan. Ecnu: Leveraging on ensemble of heterogeneous features and information enrichment for cross level semantic similarity estimation, 2014.
- [28] M. Rodriguez-Martinez. Experiences with the twitter health surveillance (ths) system. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 376–383, June 2017.
- [29] C. C. Garzon-Alfonso and M. Rodriguez-Martinez. Twitter health surveillance (ths) system. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1647–1654, Dec 2018.
- [30] Keras. Keras: The python deep learning library, 2019.
- [31] Google. Get started with tensorflow, 2019.
- [32] Apache. Apache hadoop, 2019.
- [33] Apache. Apache kafka® is a distributed streaming platform. what exactly does that mean?, 2019.
- [34] Apache. Spark overview, 2019.
- [35] Apache. Getting started with apache hive software, 2019.
- [36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [37] NVIDIA. Nvidia accelerated computing: Cuda zone, 2019.

- [38] X. Dai, M. Bikdash, and B. Meyer. From social media to public health surveillance: Word embedding based clustering method for twitter classification. In *SoutheastCon 2017*, pages 1–7, March 2017.
- [39] S. Ahuja and G. Dubey. Clustering and sentiment analysis on twitter data. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pages 1–5, Aug 2017.

Appendices

Appendix A

GitHub Repositories

The GitHub repositories of the big data and machine learning daemon are available upon request at danny.villanueva1@upr.edu. The following sections contain the links.

A.1 Big Data Platform

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

A.1.1 Machine Learning Platform

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