UNIVERSIDADE DE ÉVORA

# Automatic detection of persuasion attempts on social networks

*Rúben José Ferreira Teimas*

Tese apresentada à Universidade de Évora
para obtenção do Grau de Mestre em Engenharia Informática

Orientador *José Miguel Gomes Saias*

*Esta dissertação não inclui as críticas e sugestões feitas pelo Júri*

9 de Setembro de 2023

UNIVERSIDADE DE ÉVORA

**Escola de Ciências e Tecnologia**

Departamento de Informática

# Automatic detection of persuasion attempts on social networks

**Rúben José Ferreira Teimas**

Orientação *José Miguel Gomes Saias*

**Engenharia Informática**

Dissertação

9 de Setembro de 2023

*Esta dissertação não inclui as críticas e sugestões feitas pelo Júri*

*“What might have been lost*
*Don’t bother me”*

# Acknowledgements

As I'm finishing this chapter of my academic journey I would like to thank to all the people that throughout the years at Universidade de Évora helped grow and accomplish my goals.

First of all I would like to thank to professor José Saias, which provided me invaluable help by always pointing me in the right direction and not just letting me go after "the next big thing". I would also like to thank professor Paulo Quaresma who, although might not remember, motivated me to finish this chapter when I had doubts about it. Professor Miguel Barão who taught me (in a way that I did not find pleasurable at the time) to not settle for less when I could do more, I can finally say thank you.

To all my former colleagues at Decsis from whom I had the pleasure to share moments and learn daily, specially to Artur Romão and Paulo Caeiro who allowed me to have flexibility and take some time to progress on this work.

To all the marvelous people I've met along the way during study sessions, nights out or just chilling at Residência António Gedeão — specially to Salomé, who always believed in me even when I had some lack of faith, and Luís (Ressonha), who inspired to be who I am today and always managed to, in its own unique way, bring out the best in me — and to Beatriz, Daniel, Laura, João, Joana and Pedro, the group of people who I get to be with once a year but spend the whole year with me. I am forever grateful for your presence in my life. Without you I would probably still manage to complete it but I have to admit that would have been much harder!

At last but not least, I cannot thank enough to my family who have always been there for me and gave me all that they could, I honestly could not ask for any more. I'm happy they're here to see me thrive because part of this work is theirs.

# Contents

**Contents** ix

**List of Figures** xi

**List of Tables** xiii

**Acronyms** xv

**Abstract** xvii

**Sumário** xix

**1 Introduction** 1

1.1 Motivation . . . . . 2

1.2 Objectives and Methodology . . . . . 2

1.3 Structure . . . . . 3

**2 State of the Art** 5

2.1 Machine Learning . . . . . 5

2.1.1 Algorithms . . . . . 7

2.1.2 Problem Transformation . . . . . 9

2.2 Deep Learning . . . . . 10

2.2.1 Attention . . . . . 10

2.2.2 Network Architectures . . . . . 12

2.2.3 Transfer Learning . . . . . 14

2.3 Natural Language Processing . . . . . 15

2.3.1 Pre-processing . . . . . 15

2.3.2 Language Representation . . . . . . . . . . 16
2.4 Evaluation Metrics . . . . . . . . . . 19
2.5 Previous Approaches . . . . . . . . . . 21
2.6 Summary . . . . . . . . . . 24

**3 Problem study 25**
3.1 Data analysis . . . . . . . . . . 25
3.2 Data pre-processing . . . . . . . . . . 27
3.3 *ML* models . . . . . . . . . . 28
3.4 Summary . . . . . . . . . . 31

**4 Proposed System 33**
4.1 Architecture . . . . . . . . . . 33
4.2 Implementation and deployment . . . . . . . . . . 35
4.3 Experiment's methodology . . . . . . . . . . 36
4.4 Summary . . . . . . . . . . 36

**5 Results 37**
5.1 System evaluation . . . . . . . . . . 38
5.1.1 Baseline system . . . . . . . . . . 38
5.1.2 Text pre-processing . . . . . . . . . . 39
5.1.3 Loss function . . . . . . . . . . 40
5.1.4 DistilBERT fine-tuning . . . . . . . . . . 42
5.1.5 Hyperparameter search . . . . . . . . . . 43
5.2 Discussion . . . . . . . . . . 46

**6 Conclusion 51**
6.1 Global overview . . . . . . . . . . 51
6.2 Future Work . . . . . . . . . . 53

**A ML Models 55**

**B Taking a step back 59**
B.1 Looking at the different loss functions . . . . . . . . . . 59
B.2 Using simple DistilBERT . . . . . . . . . . 62

**Bibliography 65**

# List of Figures

2.1 Types of machine learning. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Different types of classification problems. . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.3 *ANN* example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.4 *Binary relevance example* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.5 Label Powerset example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.6 *Scaled Dot-Product Attention, adapted from [VSP+17]* . . . . . . . . . . . . . . . . . . . 11

2.7 *Multi-Head Attention, adapted from [VSP+17]* . . . . . . . . . . . . . . . . . . . . . . . 12

2.8 *LSTM* cell. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.9 Handwritten classification example with *CNN*. . . . . . . . . . . . . . . . . . . . . . . . 13

2.10 *Transformer architecture, adapted from [VSP+17]* . . . . . . . . . . . . . . . . . . . . . 15

2.11 *Word2vec architectures, adapted from [MCCD13]* . . . . . . . . . . . . . . . . . . . . . 18

2.12 *BERT single sentence classification, adapted from [DCLT18]*. . . . . . . . . . . . . . . . 19

3.1 Class distribution per split on *sub-task 1*. . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.2 Multilabel Binarizer transform. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

4.1 Final model architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

5.1 *H6* model score evolution. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

5.2 *H6* model loss evolution. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

B.1 *H6* model loss evolution, using Focal Loss (FL). . . . . . . . . . . . . . . . . . . . . . . 60

B.2 *H6* model score evolution, using FL. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60

B.3 *H6* model loss evolution, using Binary Cross Entropy (BCE). . . . . . . . . . . . . . . . 61

B.4 *H6* model score evolution, using BCE. . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

B.5 Plain DistilBERT model loss evolution. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
B.6 Plain DistilBERT model score evolution. . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
B.7 Plain DistilBERT model, with 5 layers frozen, loss evolution. . . . . . . . . . . . . . . . . . 63
B.8 Plain DistilBERT, with 5 layers frozen, model score evolution. . . . . . . . . . . . . . . . . 63

# List of Tables

2.1 Example binary confusion matrix. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.1 Statistics about the persuasion techniques. . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.2 Corpus dimension before and after pre-processing . . . . . . . . . . . . . . . . . . . . . 28

3.3 Results using TF-IDF as feature extraction technique. . . . . . . . . . . . . . . . . . . . 29

3.4 F1-Score by class on each model using TF-IDF. . . . . . . . . . . . . . . . . . . . . . . . 29

3.5 Results using Word2vec as feature extraction technique. . . . . . . . . . . . . . . . . . . 30

3.6 F1-Score by class on each model using Word2Vec. . . . . . . . . . . . . . . . . . . . . . 30

5.1 Baseline system hyperparameters. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

5.2 F1-Score for each class on final architecture's baseline model. . . . . . . . . . . . . . . . . 39

5.3 F1-Score for each class on using text pre-processing on final architecture's baseline model. 40

5.4 F1-score for each class using different loss functions. . . . . . . . . . . . . . . . . . . . . 41

5.5 Micro and Macro F1-scores for different loss functions. . . . . . . . . . . . . . . . . . . . 42

5.6 Macro and Micro F1-Score freezing DistilBERT's layers. . . . . . . . . . . . . . . . . . . 42

5.7 Search space for hyperparameters. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

5.8 Models trained using Tree Parzen Estimator (TPE) as search algorithm. . . . . . . . . . . 45

5.9 Comparing the final models results by class. . . . . . . . . . . . . . . . . . . . . . . . . . 47

5.10 SemEval's systems comparison. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

A.1 Precision by class on each model using TF-IDF. . . . . . . . . . . . . . . . . . . . . . . . 56

A.2 Recall by class on each model using TF-IDF. . . . . . . . . . . . . . . . . . . . . . . . . 56

A.3 Precision by class on each model using Word2Vec. . . . . . . . . . . . . . . . . . . . . . . 57

A.4 Recall by class on each model using Word2Vec. . . . . . . . . . . . . . . . . . . . . . . . 57

# Acronyms

| | |
|---|---|
| **ANN** | Artificial Neural Network |
| **ASL** | Asymmetric Loss |
| **BR** | Binary Relevance |
| **BCE** | Binary Cross Entropy |
| **BoW** | Bag of Words |
| **CNN** | Convolutional Neural Network |
| **CV** | Computer Vision |
| **DL** | Deep Learning |
| **FL** | Focal Loss |
| **LLM** | Large Language Model |
| **LP** | Label Powerset |
| **LSTM** | Long Short-term Memory |
| **ML** | Machine Learning |
| **NLP** | *Natural Language Processing* |
| **PT** | Problem Transformation |
| **RL** | Reinforcement Learning |
| **RNN** | Recurrent Neural Network |
| **SGD** | Stochastic Gradient Descent |
| **SMBO** | Sequential model-based optimization |
| **SL** | Supervised Learning |
| **TL** | Transfer Learning |
| **TPE** | Tree Parzen Estimator |
| **UL** | Unsupervised Learning |

# Abstract

The rise of social networks and the increasing amount of time people spend on them have created a perfect place for the dissemination of false narratives, propaganda, and manipulated content. In order to prevent the spread of disinformation, content moderation is needed, however it is unfeasible to do it manually due to the large number of daily posts. This dissertation aims at solving this problem by creating a system for automatic detection of persuasion techniques, as proposed in a SemEval challenge. We start by reviewing classic machine learning and natural language processing approaches and go through more sophisticated deep learning approaches which are more suited for this type of complex problem. The classic machine learning approaches are used to create a baseline for the problem. The architecture proposed, using deep learning techniques, is built on top of a DistilBERT transformer followed by Convolutional Neural Networks. We study how our usage of different loss functions, pre-processing the text, freezing DistilBERT layers and performing hyperparameter search impact the performance of our system. We discovered that we could optimize our architecture by freezing the two initial DistilBERT's layers and using asymmetric loss to tackle the class imbalance on the dataset presented. This study resulted in three final models with the same architecture but using different parameters where the first showed signs of overfitting, one did not show sings of overfitting but did not seem to converge and other seemed to converge but yielded the worst performance of all three. They presented a micro f1-score of 0.551, 0.526 and 0.509 and were placed in 3rd, 6th and 11th place respectively in the overall table. The models can only classify textual elements as the multimodal component is not implemented on this iteration but only discussed.

**Keywords:** Machine Learning, Natural Language Processing, Deep Learning, Persuasion Analysis, Social Networks

# Sumário

## Deteção automática de tentativas de persuasão em redes sociais

O crescimento das redes sociais e o aumento do tempo que as pessoas passam nelas criaram um lugar perfeito para a disseminação de falsas narrativas, propaganda e conteúdo manipulado. Para evitar a disseminação da desinformação, é necessária a moderação do conteúdo, porém é inviável fazê-la manualmente devido ao grande número de conteúdo diário. Esta dissertação visa resolver este problema através da criação de um sistema de deteção automática de técnicas de persuasão, conforme proposto num desafio da SemEval. Começamos por rever as abordagens clássicas de aprendizagem automática e processamento de linguagem natural, passamos de seguida por abordagens mais sofisticadas de aprendizagem profunda que são mais adequadas para esse tipo de problema complexo. As abordagens clássicas de aprendizagem automática são usadas para criar um ponto de partida para o problema. A arquitetura proposta, utilizando técnicas de aprendizagem profunda, é construída sobre um transformer DistilBERT seguido de redes neuronais convolucionais. Estudamos de que forma o uso de diferentes funções ativação, pré-processamento do texto, congelamento de camadas do DistilBERT e realização de pesquisa de hiperparâmetros afetam o desempenho do nosso sistema. Descobrimos que poderíamos otimizar nossa arquitetura congelando as duas camadas iniciais do DistilBERT e usando asymmetric loss para lidar com o desequilíbrio de classes no conjunto de dados apresentado. Este estudo resultou em três modelos finais com a mesma arquitetura, mas usando parâmetros diferentes, onde o primeiro mostrou sinais de overfitting, um não mostrou sinais de overfitting mas não parece convergir e outro parece convergir, mas produziu o pior desempenho de todos os três. Apresentaram micro f1-score de 0.551, 0.526 e 0.509 e ficaram em 3º, 6º e 11º lugares, respectivamente, na tabela geral. Os modelos podem apenas classificar elementos textuais, pois o componente multimodal não é implementado nesta iteração, mas apenas discutido.

**Palavras chave:** Aprendizagem Automática, Processamento de linguagem natural, Aprendizagem profunda, Análise de persuasão, Redes Sociais

# 1 Introduction

With the appearance of *Web 2.0* regular internet users were able to upload their own content to the web by using platforms such as blogs and social networks, amongst them: *Facebook*, *Twitter*, *Instagram* and *Reddit*. Throughout the years time spent by users on those platforms has increased. It is reported that, on average, each American spent more than 1300 hours on social networks during the year of 2021. [Suc21]

Whilst allowing users to post as they are pleased increases the diversity of content which, at first sight, is good for free speech it also comes with the risk of having undesired content. People that post undesired content are often referred as *trolls*, usually users with anonymous identities who use techniques such as *name calling*, *whataboutism*, presenting irrelevant data (*red herring*), amongst others.

*Troll*'s existence can be perceived as a minor inconvenient or even innocuous, however they can play a big part on real world events. An well known case is the possible interference of Russian *trolls* on the *USA*'s 2016 Presidential Elections. [Sch20] They used *memes* as their primary weapon.

A *meme* can be defined, in a broad sense, as an image with text on top of it usually of humorous nature. By using this type of media Russian *trolls* were able to manipulate public opinion and sabotage political debates. [Sch20]

## 1.1 Motivation

Given the amount of information available today on the web, the societal challenge relative to scarcity of information has been overcome (in certain regions of the world) lending its place to new challenges like information filtering, unverified sources of information and the information's fast consumption. The latter somehow strengthen the first challenge since users spend less time doing research which can lead to misinformation consumption.

These challenges could be fought by having regulating entities verifying the content validity and its motivations but the amount of information makes this solution unfeasible, which turns said challenges into perfect candidates to be played by "intelligent" systems. Some companies, like *Meta* (previously known as *Facebook*), already use [Met] these kind of systems to evaluate content, in this case, to detect hate-speech.

By the end of this dissertation we aim at an "intelligent" system capable of identifying persuasion techniques. To achieve said goal we'll try to solve 2 tasks, announced on a *SemEval* challenge [Sem], with different modal properties. The tasks are particularly interesting since they go from plain text classification to image+text classification.

The final result can later be integrated into an array of applications making an modular piece of software that can be reused according to each needs.

## 1.2 Objectives and Methodology

The work's main objective is to create a proper model for 2 of SemEval 2021 Task 6 challenges [Sem] on "detection of persuasion techniques in texts and images" sub-tasks. The sub-tasks revolve around the same data with subtle differences both on the dataset and the approach. There are 2 of them, with one being a multimodal problem:

- **Sub-task 1**:

  Given the textual content of a *meme*, identify which persuasion techniques (out of 20 possible ones) are used in it. This is a multilabel classification problem.

- **Sub-task 3**:

  Given a meme, identify which persuasion techniques (out of 22 possible ones) are used in the meme, considering both the text and the image. This is a multilabel classification problem.

Building good models requires not only a good knowledge about the problem but also about the datasets and the many techniques that can be used on top of them. That said, it can be achieved by splitting it in smaller tasks like:

- **Study the data**:

  An important step in classification problems is to understand the data. By analysing the datasets we can gather more information about its nature, how the data is structured, which classes appear most times and other specificities.

- **Explore *Natural Language Processing* (NLP) and Machine Learning (ML) techniques**:

  Textual elements will be used on all sub-taks, therefore it is important to that have a good knowledge about the possible techniques to pre-process the text before being fed to a model. There are also plenty of papers which describe their approach ti similar tasks, therefore it is useful to read them and learn from them.

- **Explore Deep Learning (DL) models**:

  Even though certain multilabel tasks can be solved using simple *ML* tasks, most of them are solved nowadays using solutions such as *transformers* which have gained more and more popularity nowadays. Such architectures are used on the papers submitted by *SemEval* participants.

- **Develop the data-pipeline and the models**:

  With a broader knowledge about the topics above we need to start thinking about the models architecture. Multiple experiments should be made with different models and techniques.

- **Evaluate the results**:

  After developing and training the models the results must be compared with the ones presented on the challenge's paper [DAS+21]. The results must be interpreted.

This should be an iterative process where models are implemented and trained as our knowledge about theoretical topics expand.

## 1.3 Structure

This dissertation is structured in 6 chapters, with this chapter serving as an introduction to the problem.

In Chapter 2 we explain concepts used throughout this work and present other author's solutions to somehow similar works.

Chapter 3 contains the problem's data analysis and some initial experiments on the data.

Chapter 4 describes the proposed solution for the sub-tasks as well as its implementation and deployment.

In Chapter 5 the experiments and its the results for the trained models are presented and compared with the ones from the original paper.

Chapter 6 addresses what we've accomplished with this work and its future improvements.

# 2

# State of the Art

Detection of propaganda and hate speech on text has been a field of interest for quite a while with multiple academic and *R&D* papers on it. At the time of writing more and more multimodal tasks concerning hate speech detection have been released with respective approaches.

In the following sections the state of the art of our problem is presented by describing important *NLP*, *ML* and *DL* concepts and having in mind the approaches used by other with similar tasks.

## 2.1 Machine Learning

Machine Learning is a sub-field of Artificial Intelligence (*AI*) that "enables computers to think and learn on their own" [ANK18]. Without having explicitly programmed instructions, the computers learns by looking at different data and modifying their actions in order to maximize the correctness of its output.

Simply put "A computer program is said to learn from experience $E$ with respect to some task $T$ and some performance measure $P$, if its performance on $T$, as measured by $P$, improves with experience $E$" [Mit97].

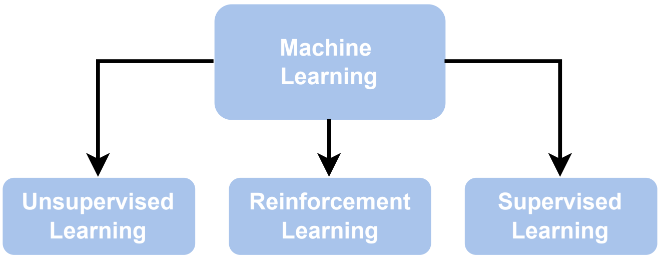


Figure 2.1: Types of machine learning.

Within *ML* there are 3 basic approaches: *Unsupervised Learning*, *Reinforcement Learning* and *Supervised Learning*.

When the model is not given any set of labels on the training data it must find patterns or some data's structure, this is called Unsupervised Learning (UL). *Clustering* is an *UL* technique and its goal is to group similar data points together in a way that data points in the same group (cluster) are more similar to each other than to those in other clusters [ANK18]. Another example of *UL* is *Anomaly Detection* that aims at finding data points which are inconsistent (outliers) with the remaining set [ANK18].

Reinforcement Learning (RL) algorithms are used when a decision is to be made based on past experiences of learning. The machine agent learns the behaviour using trial and error sort of interaction with the continuously changing environment. It provides a way to program agents using the concept of rewards and penalties without specifying how the task is to be accomplished [ANK18]. Game playing programs and automated trading systems are some popular examples.

In Supervised Learning (SL) we train an *ML* algorithm with a labeled dataset, where the correct output/set of label is provided for a given input. The main objective of supervised learning is to find a mapping function from input variables ($X$) to output variable (Y) [ANK18]. *SL* problems are categorized into *Regression* and *Classification* problems. In a *Regression* problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a *classification* problem, we are instead trying to predict results in a discrete output.

There are different types of classification problems as illustrated on figure 2.2, however not all algorithms are suitable for them.

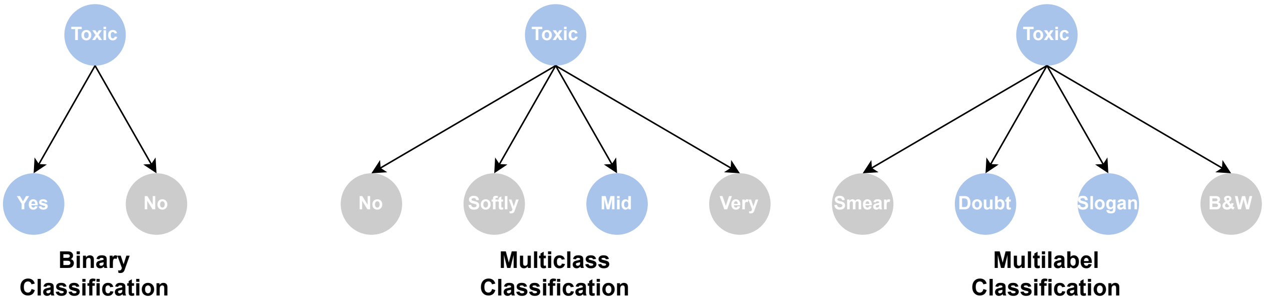


Figure 2.2: Different types of classification problems.

While on *binary* and *multiclass* classification each data point can only be classified as 1 class out of the available disjointed set of classes, 1 in 2 or 1 in $N$ respectively, on *multilabel* classification a data point can be classified with 0 to $N$ classes out of the $N$ classes available.

### 2.1.1 Algorithms

#### Decision Tree

Decision Tree is a supervised learning algorithm often used in classification and regression tasks. The algorithm creates a model based on decision rules inferred from the data features.

To create the models, the features that contain the most information about the target feature are located and then split the dataset along with their values. There are multiple measure to compute the best split, such as: *Gini impurity* and *Information Gain* [Kot13].

The one used by default on the *sklearn* implementation is the *Gini Impurity*, which is given by the formula below.

$$Gini(D) = 1 - \sum_{j=1}^{n} {p_j}^2 \tag{2.1}$$

Where $p_j$ is the relative frequency of class $j$ in the dataset (partition) $D$.

The models generated by decision trees are usually easy to comprehend and visualize [Kot13] but very prone to overfitting when dealing with high dimensional data. There are strategies to prevent the model from overfitting such as *pruning* and checking for *data imbalance*.

#### K Nearest Neighbors

K Nearest Neighbors (*KNN*) is a simple, yet flexible, algorithm that can be applied either in unsupervised (as clustering) and supervised (classification and regression) learning.

When used in supervised learning, given a entry $p$ and $N$ training instances the algorithm identifies $k$ nearest neighbors and classifies $p$ with the same class of the majority of $k$. The distance $d$ between the points $a$ and $b$ is calculated using a distance function such as the *Euclidean Distance*.

$$d(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{2.2}$$

Where $i$ is the number of features of the points.

#### Naive Bayes

Based on *Baye*'s theorem (2.3), it is called *naive* because it assumes the feature's mutually independence (2.4). This independence assumption is often violated, but *Naive Bayes* classifiers still tend to perform very well under this unrealistic assumption [Ras14], as long as the features are not strongly correlated.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

$$P(A,B) = P(A)P(B) \tag{2.4}$$

With equation 2.3 we're trying to find the probability of event *A* occurring, given that event *B* is true. *B* is also know as the **evidence** [Ras14]. When applied to a dataset, *P(A/B)* would be the probability of class *A* being predicted knowing feature *B* occurred.

### Artificial Neural Networks

The concept of Artificial Neural Network (ANN) is very popular nowadays when performing classification tasks. The name stems from the idea of a structure that would work like the human brain, where connect neurons pass information to each other outputting a desired result [AJO$^+$18].

An *ANN* is composed by multiple layers, each of them with an parameterizable number of neurons. A neuron can be defined as set of inputs *x*, a set of weitghs *w*, a bias *b* and an activation function *f*. These inputs are converted by the neuron into a single output that can later be used as input by another layer of neurons.

The function $f(x_1, x_2, \ldots, x_n)$ can be expressed as a weight sum, although it may vary from network to network:

$$f(x_1, x_2, \ldots, x_n) = b + \sum_{i=1}^{n} w_i x_i \tag{2.5}$$

The most basic form of an *ANN* contains only a hidden layer as well as the input and output layers, the number of neurons at each layer can vary. This is also known as a *shallow neural network*.

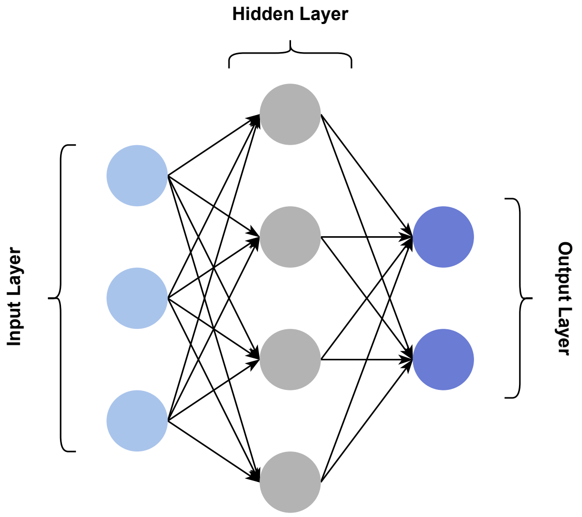


Figure 2.3: *ANN* example.

During training, after computing an output $\hat{y}$ the *ANN* will use the true value $y$ to calculate how far the prediction is from reality, and to do so a *loss* function is used. Gradient methods are then applied in order to minimize the loss and they will be back-propagated to the network in order to updated the weights [Bod02].

There are different types of neural networks with different configurations and properties from which we can take advantage to tackle different problems.

Long Short-term Memory (LSTM) networks are very popular in tasks concerning *NLP* and time-series [SJ19]. For image classification or object detection, Convolutional Neural Network (CNN) are the most common choice [AG17].

### 2.1.2 Problem Transformation

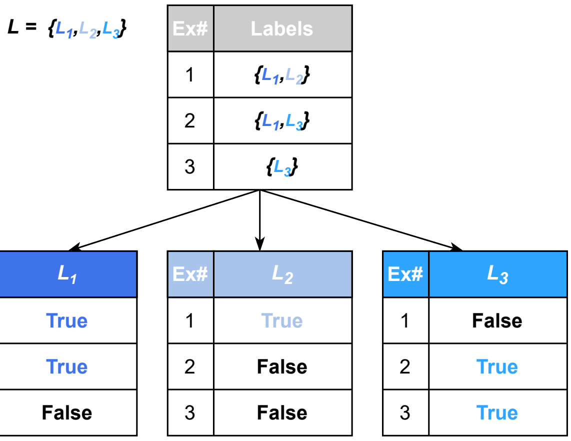
There are a reduced number of classifiers that support multi-label classification, due to the original algorithm's nature. Problem Transformation (PT) turns *multilabel* classification problems into either *binary* or *multiclass* problems. Some well known techniques are Binary Relevance (BR) and Label Powerset (LP). [CMM11].

#### Binary Relevance

Given a classifier $c$ and problem with $n$ classes, where $n > 2$, $n$ single-class binary classifiers $C_i, i = 1, \ldots, n$ are created as illustrated on figure 2.4.

The BR initially transforms the original dataset $D$ into $n$ binary datasets $D_{y_j}, j = 1, \ldots, n$, where each $D_{y_j}$ contains all the examples of $D$, but with with a positive or negative class related to the single class $y_j$. This new datasets are then fed to the $C_i$ classifiers [CMM11].

The prediction output is the union of all classes per classifier.



L = {$L_1$,$L_2$,$L_3$}

| Ex# | Labels |
|---|---|
| 1 | {$L_1$,$L_2$} |
| 2 | {$L_1$,$L_3$} |
| 3 | {$L_3$} |

| Ex# | $L_1$ |
|---|---|
| 1 | True |
| 2 | True |
| 3 | False |

| Ex# | $L_2$ |
|---|---|
| 1 | True |
| 2 | False |
| 3 | False |

| Ex# | $L_3$ |
|---|---|
| 1 | False |
| 2 | True |
| 3 | True |

Figure 2.4: *Binary relevance example*

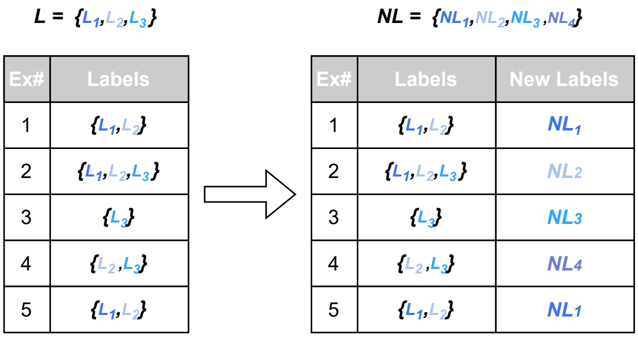
Compared to other multilabel classification techniques, it offers a low computational complexity and it is easy to parallelize since each classifier is trained independently. Assuming a classifier $c$ is bound to a complexity of $O(C)$ and there are $n$ classes, the complexity would be $n \times O(C)$.

This technique makes the assumption of class independence.

#### Label Powerset

This method transforms the multilabel problem into one single-label multiclass classification problem, where the possible values for the transformed class attribute is the set of distinct unique subsets of labels present in the original dataset [CMM11]. Figure 2.5 illustrates this idea.

This approach does not assume the labels are independent, nonetheless the computational complexity



$L = \{L_1, L_2, L_3\}$

| Ex# | Labels |
|---|---|
| 1 | $\{L_1, L_2\}$ |
| 2 | $\{L_1, L_2, L_3\}$ |
| 3 | $\{L_3\}$ |
| 4 | $\{L_2, L_3\}$ |
| 5 | $\{L_1, L_2\}$ |

$NL = \{NL_1, NL_2, NL_3, NL_4\}$

| Ex# | Labels | New Labels |
|---|---|---|
| 1 | $\{L_1, L_2\}$ | $NL_1$ |
| 2 | $\{L_1, L_2, L_3\}$ | NL2 |
| 3 | $\{L_3\}$ | NL3 |
| 4 | $\{L_2, L_3\}$ | NL4 |
| 5 | $\{L_1, L_2\}$ | NL1 |

Figure 2.5: Label Powerset example.

tends to be higher. Given a training set with $n$ samples and $k$ labels the computational complexity of Label Powerset is upper bounded by $min(n, 2^k)$ [Sor10].

Another drawback from this technique is that it cannot predict unseen labels. If the combinations of labels has never been observed in training the generated model will never be able to predict it. This can cause overfitting.

## 2.2 Deep Learning

Deep learning is a sub-field of *AI* that uses *ANN* with many layers, thus the term *deep*. In general, *DL* approaches require more data to train the models than the ones using machine learning. *ANN* are trained iteratively which allows them learn from their own experience, using *loss functions* and *backpropagation* [DCWZ16].

In the last few years it has been successful in a wide range of applications, including *speech recognition*, *natural language processing* and *computer vision*. The concepts bellow helped establish *deep-learning* as a *go-to* approach for those problems.

### 2.2.1 Attention

Attention can be defined as a mapping between a query ($Q$) and a set of key-value pairs ($K$,$V$) to an output ($O$). The output ($O$) is computed as a weighted sum of the values ($V$) where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [VSP$^+$17].

**Scaled Dot-Product Attention**

The authors of [VSP$^+$17] proposed a new approach to attention focused on the dot-product operation. The input consists of queries and keys of dimension $d_k$, and values of dimension $d_v$. The attention is computed by first applying the dot-product between all the queries and the keys, which will be divided by $\sqrt{d_k}$. The result is then passed to a softmax function to obtain the weights of the values.

Queries, keys and values are put into matrices $Q$, $K$ and $V$, respectively. The output is given by the function:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.6}$$

The attention function is identical to the dot-product attention, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. The scaling factor is crucial when the dot-product between $Q$ and $K$ grows large in magnitude, as this values will be fed to the softmax function producing extremely small gradients. The scaling factor will counteract this effect.

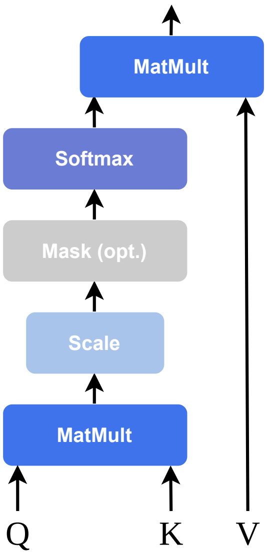


Figure 2.6: *Scaled Dot-Product Attention, adapted from [VSP+17]*

### Multi-Head Attention

While the previous section showed a way to calculate attention with a single function it may be beneficial for the system to allow our attention mechanism to jointly use different representation of the subspace of $Q$, $K$ and $V$.

This can be achieved by linearly projecting the queries, keys and values $h$ times, with different learned projections to $d_k$, $d_k$ and $d_v$, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding $d_v$-dimensional output values. These are concatenated and once again projected, resulting in the final values.

Multi-head attention is expressed as a function of:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(head_1,...,head_h)W^O$$

$$\textbf{where } head_i = \text{Attention}(QW_i^Q,\ KW_i^k,\ VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$.

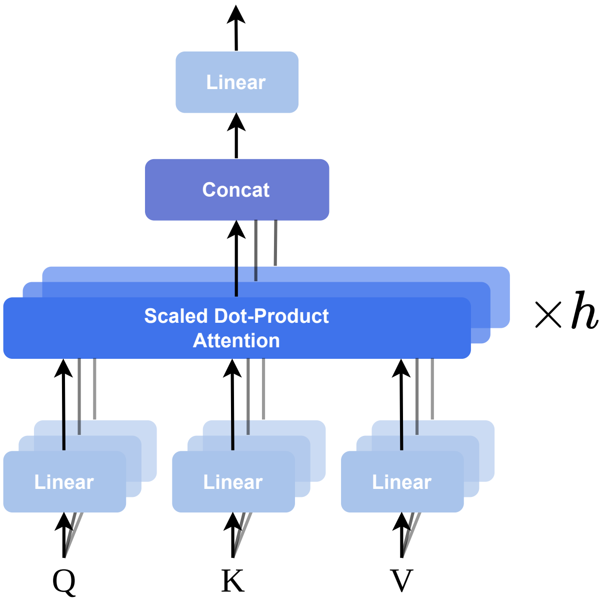


Figure 2.7: *Multi-Head Attention, adapted from [VSP+17]*

## 2.2.2 Network Architectures

### LSTM

Traditional neural networks are not well-suited to processing sequential data, such as *time-series* and *natural language*, because they assume that the input data is independent which won't allow them to capture the effect of past inputs.

Because of that, Recurrent Neural Network (RNN) were introduced. An RNN maintains an internal state memory based on previous information (*short-term memory*). The hidden state is then updated based on the input and the previous hidden state stored in memory [HXY15].

When dealing with long sequences *RNN*'s performance start to resent due to what's called *vanishing gradient*. As the gradients are propagated through the layers, the gradients start to get more and more diluted, turning it residual. The network will not be able to learn properly.

*LSTM*s can overcome this problem by adding gates which are units within the LSTM that control the flow of information into and out of the cell state. These gates control how memory is affected by the current and previous inputs, they also control the output of the current memory cell.

A *LSTM* still work like a regular *RNN* except that the hidden layer updates are replaced by purpose-built memory cells [HXY15].

The memory is updated at each instant using the the following rules:

$$i_t = \sigma(W_i x_t + V_i h_{t-1}) \tag{2.7}$$

$$f_t = \sigma(W_f x_t + V_f h_{t-1}) \tag{2.8}$$

$$o_t = \sigma(W_o x_t + V_o h_{t-1}) \tag{2.9}$$

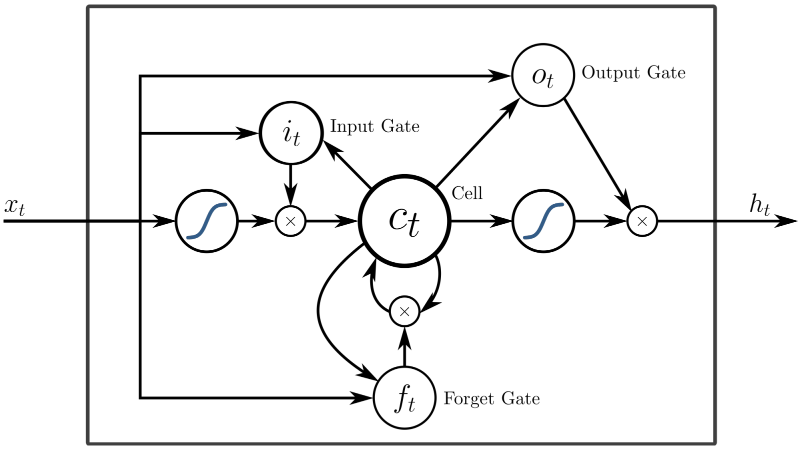


Figure 2.8: *LSTM* cell.

$$c_t = f_t \cdot c_{t-1} + i_t \cdot tanh(W_c x_t + V_c h_{t-1}) \tag{2.10}$$

$$h_t = o_t \cdot tanh(c_t) \tag{2.11}$$

Where $i_t$ is the input gate, $f_t$ the forget gate, $o_t$ the output gate, $c_t$ the memory cell and $h_t$ the hidden state. The symbols $W$ and $V$ represent weight matrices that are learnt and shared across all time steps.

### CNN

Convolutional Neural Networks were first introduced in [LBBH98] as a *state of the art* approach for handwritten character recognition. This *ANN* architecture relies on 2 operations in its hidden layers: *convolution* and *pooling*.

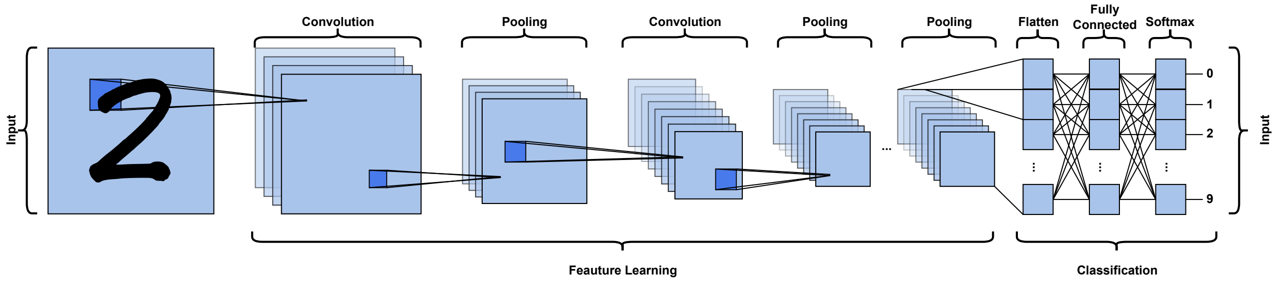


Figure 2.9: Handwritten classification example with *CNN*.

The convolution is a mathematical operation of two functions ($f$ and $g$) that produces a third function ($f * g$) that expresses how the shape of one is modified by the other. It is given by:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau \tag{2.12}$$

It is a common technique in Computer Vision (CV) as it can be used to perform tasks such as image smoothing and edge detection. Convolution is often performed in *CV* as it follows: given an input image $I$ and a kernel $k$, we apply (slide) $k$ throughout $I$. The mathematical formulation is similar to 2.12 with some minor changes:

$$(I * k)(x, y) = \int_{-\infty}^{\infty} I(x - i, y - j) \cdot k(i, j) didj \tag{2.13}$$

The result of the integral applied to every pair of points $(x, y)$ will be a feature map. Each kernel is responsible by identifying a feature, therefore the convolution will be applied $N$ times (one for each kernel) and $N$ feature maps will be generated. This process is the behaviour of a Convolution Layer.

Usually, every convolution layer is followed by a Pooling Layer. Pooling is a down-sampling technique applied to a feature map in order to decrease its dimensions while preserving the most important features. The most used pooling variations are Max Pooling and Average Pooling. They are very straightforward, the first variation takes the highest value of the feature map while the second approach does an arithmetic mean with all the values of the feature map.

Despite being very popular in *CV*, convolutional networks have also been used in text classification [CDY+21] [ZWZ21]. They are known as *Text-CNN* and follow the same structure of a regular *CNN* but instead of using an input image it takes a matrix of word embeddings.

#### Transformer

The paper "Attention Is All You Need" [VSP+17] presented a new network architecture: the *Transformer*. It relies solely on attention mechanism, dismissing any recurrence and convolutions. By eliminating the sequential factor the architecture can be paralleled making it faster to train the models.

The *Transformer* follows an encoder-decoder structure, where the encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations **z** $= (z_1, \ldots, z_n)$. Base on representations **z**, the decoder then generates an output of sequence $(y_1, \ldots, y_m)$ of symbols one element at the time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next [VSP+17]. Additionally, the Transformer takes advantage of positional encoding, self-attention mechanisms and point-wise, fully connected layers. These layers are present on both encoder and decoder.

The encoder can be composed of multiple stacks, where each layer has two sub-layers: the first being a multi-head self attention, and the second a position-wise fully connected feed-forward network.

The decoder is similar to the encoder but it adds a starting sub-layer of masked multi-head attention. This extra sub-layer with masked attention prevents the leftward information flow in the decoder, preserving the auto-regressive property.

### 2.2.3 Transfer Learning

Transfer Learning (TL) is a technique in which a model trained on a set of data is used as a starting point for a second model with different data. These two datasets are usually of the same domain or, at least, somehow related. Otherwise there will be no useful information to transfer. [ZQD+19]

The process of making *TL* consists essentially on re-using the initial model's trained layers and the respective weights. These weights can either be frozen, in which the weights will be untouched during the fine-tuning phase for the new data, or they can be flexible and changed during the training process. [BKD21]

Using *TL* can sometimes mitigate the effects of having a small dataset for a specific task. Many areas

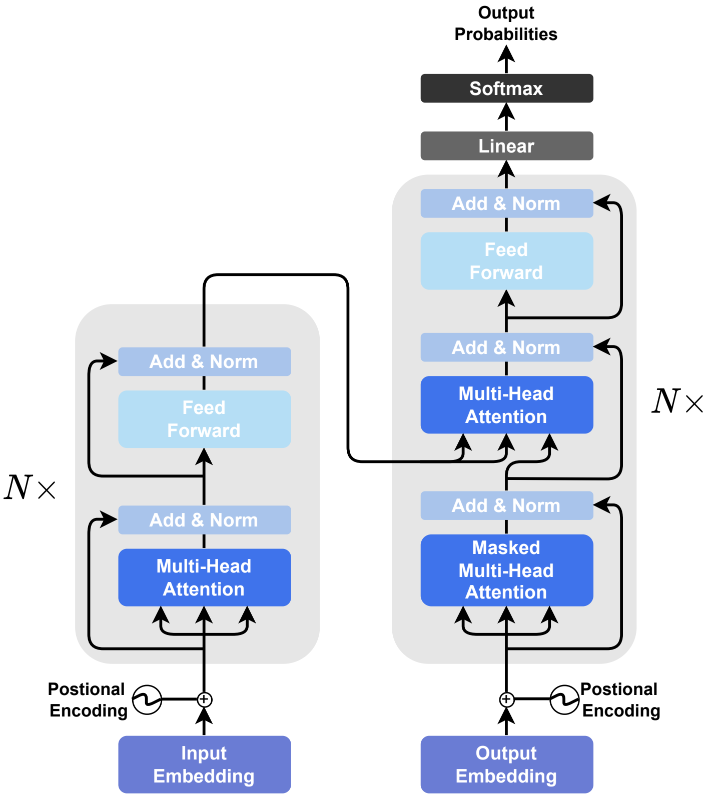


Figure 2.10: *Transformer architecture, adapted from [VSP+17]*

have benefited from using it, specially *NLP* with the appearance of Large Language Model (LLM) such as *BERT* [DCLT18] or *ULMFiT* [HR18].

## 2.3 Natural Language Processing

Natural language processing (*NLP*) is a sub-field of artificial intelligence and linguistics that focuses on the interaction between computers and humans in the form of natural language [KSND21]. It involves developing algorithms and models that can process, analyze, and generate human language. *NLP* is used in a variety of applications, such as information retrieval, machine translation, and text classification.

### 2.3.1 Pre-processing

Each language has its own words and structure but all of them share the use of punctuation, the existence of verb tenses, pronouns and other parts of speech. These elements give humans cues and context in order to have a better understanding. They are not as useful to computers, in fact, some of these elements tend to generate noise when training a model thus the need of pre-processing.

There are multiple pre-processing techniques that can be used according to each problem's needs, they are usually chained into a pipeline to process the raw-data. Some of the most common techniques are explained bellow.

#### Tokenization

The process of tokenization consists of breaking a stream of text into segments. These segments, also known as *tokens*, can be characters, words or even full sentences.

#### Stop word removal

In a language there are a lot of words that do not add much meaning to a sentence such as “a”, “the” or “and”. These words are usually called *stop words*. Most of the times it is useful to remove these words because, even if they don’t add any noise to the training process, they increase our *corpus* dimension.

#### Text normalization

Small variations of the same word can affect the training process, to prevent it there are multiple things that can be done. They can be as simple as turning all the text to lower case or replacing an acronym for its regular word and go to as complex as to reduce a word to its base form. The latter can be achieved with either *Stemming* or *Lemmatization*.

*Stemming* tries to reduce a word to its base form by chopping either its prefix or suffix. It does not guarantee the stem is the root form of the word. For instance, given the word *studied*, the suffix is *-ed* which would produce the stem *studi* [KSND21], which is not a valid word.

On the other hand, *Lemmatization* takes into account the part of speech in order to obtain the correct root form, in this case called *lemma* [KSND21]. For this reason, *Lemmatization* can produced a more precise text normalization at the expense of additional computational resources.

### 2.3.2 Language Representation

A language is essentially a known communication system between two parties defined by a set of rules. Humans, usually, communicate with each other using sounds and words that are decoded when heard/read. Computers however cannot decode words in the way they are normally encoded, and that’s when language representation comes into play.

There are multiple ways to represent text, with some of them using simply statistical elements while others use more advanced techniques. These representations are frequently classified as *discrete* or *continuous*.

A text representation technique is *discrete* when the words are considered to be mutually independent. In order words, they do not express any relationship between words. One the same line, a *continuous* text representation represents the words in such a way that there is a relationship between them.

Bellow we present some common language representation techniques.

#### Bag of Words

Bag of Words (BoW) is one of the simplest way to represent text. The distinct words are given and index which will store the computed frequency of occurrence of that word.

Given a vocabulary $V = \{w_1,w_2,...,w_n\}$ with $n$ words and a document $D$ with $m$ words, it can be expressed as a vector $x$, in which the $x_i$ position represents the number of times the word on position $i$ occurs.

This technique is sometimes chosen due to its low computational complexity but its simplicity makes the representation less robust. It does does preserve the words order nor its importance. For instance, if the *stop words* are not removed from the text they will be considered the most important features they appear more frequently.

### TF-IDF

This representation improves over the previous one by taking into account both the frequency of the word in the document and the frequency of the word in the entire corpus. Recurrent words in a document but not in the corpus as a whole are given a higher weight, while words that appear frequently in the corpus but not in the document are given a lower weight.

The term frequency can be calculated similarly to the *BoW*, given a term $w$ and a document $d$ in which:

$$\text{TF}(t, d) = count(t, d) \tag{2.14}$$

The inverse document frequency expresses the importance of a term $t$ across the space of documents $D$ and is given by:

$$\text{IDF}(t, D) = \log\left(\frac{N}{|d \in D : t \in d|}\right) \tag{2.15}$$

in which $N$ is the total number of documents in $D$, and the expression on the denominator expresses the number of documents and in which the term appears.

Therefore, the *TF-IDF* is given by:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D) \tag{2.16}$$

By doing some minor tweaks over the *BoW* representation *TF-IDF* tends to show better results, as shown by [ALA22].

### Word2vec

Introduced in [MCCD13] this language representation captures both syntactic and semantic similarities between words.

Instead of being a purely statistical model this representation makes usage of an *ANN* and there are two possible architectures: *Continuous Bag-of-Words* (*CBoW*) and *Skip-gram*.

Both architectures require us to define a sliding window of size $N$ where the $N$ words before and after the word $w$ will be useful. The words on the sliding window are called *context words* and can be defined as a group $C$, where $C = \{w_{t-N},...,w_{t+N}\} \setminus \{w_t\}$.

*CBoW* architecture allow us to, given $C$, predict the target word ($w_t$). The training of the network will end up generating the word's representation which will be stored on the weight matrix ($W_o$) connecting the hidden layer to the output layer. The $i_{th}$ row of $W_o$ represents the $i_{th}$ word of the vocabulary.

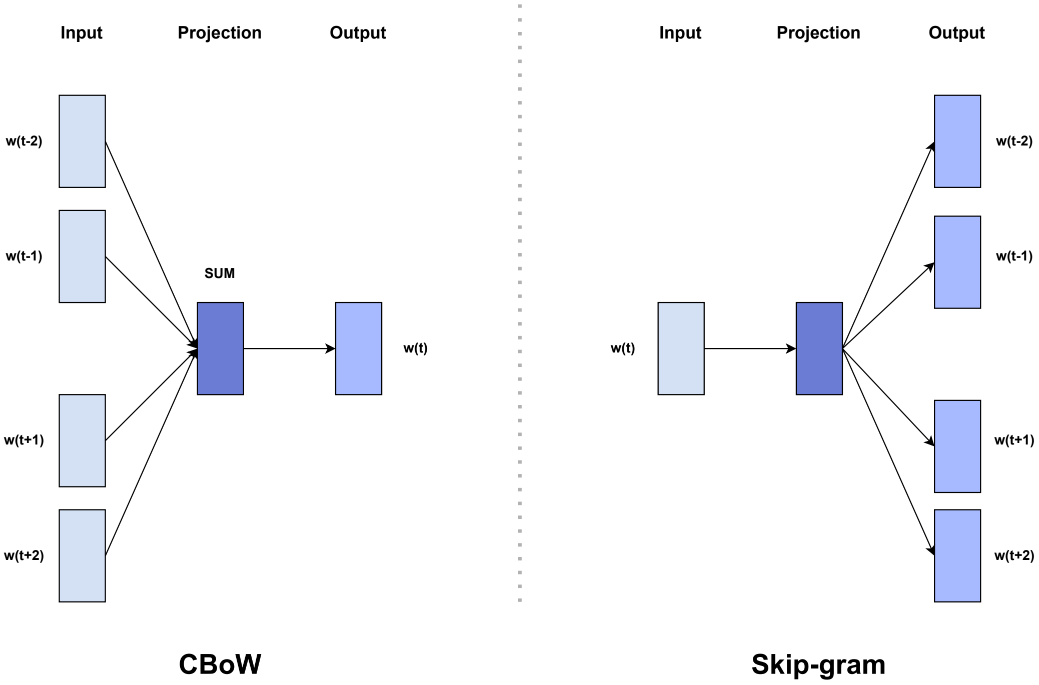


Figure 2.11: *Word2vec architectures, adapted from [MCCD13]*

The *Skip-gram* architecture mirrors the *CBoW* and instead of taking the context-words as input it tries to predict them. The learn representation is stored on the matrix $W_i$, between the input layer and the hidden layer.

Training a model like *Word2vec* is computationally more expensive than using any of the previous representations, especially the *Skip-gram* architecture which requires a larger amount of data in order to converge. There is a clear trade-off between efficiency and a richer representation (*word embeddings*) of the vocabulary. Even though the vocabulary representation is richer it is still context free, which means that homonyms will only have one representation [STF21].

### BERT

*BERT* (Bidirectional Encoder Representations from Transformers) [DCLT18] is a stack of bidirectional encoders based on the original *Transformer* [VSP$^+$17] architecture. The architecture has two steps: the *pre-training* and *fine-tuning*. During *pre-training* the model is trained on unlabeled data over different pre-training tasks [DCLT18], which will result in a *pre-trained* language model (*LM*). For the *fine-tuning* step, the *LM*'s parameters will be used as a starting point to specific tasks which will be trained on the task's data.

A major concept of BERT's representation is the bidirectionally which means that both the previous and next words determine the representation of a word. Let's take for example the word "bank", given the sentences "I am going to the bank to sit down" and "I am going to the bank to withdraw cash" the meaning of the word "bank" is easily inferred by the words "sit" and "withdraw" respectively, however if the model was unidirectional only the previous words to "bank" would affect the representation. [STF21] The bidirectionally helps the representation to have an even stronger context-aware component.

Bidirectional representation is achieved by masking some percentage of the input randomly, and then

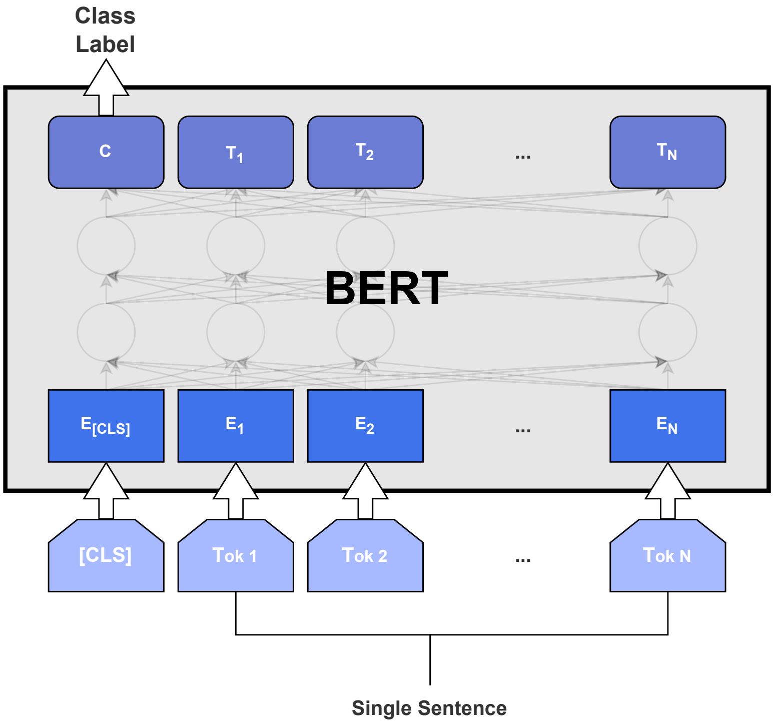


Figure 2.12: *BERT single sentence classification, adapted from [DCLT18].*

predicting the masked token. The authors called this procedure as a *masked language model* (*MLM*). The masked tokens are replaced by the special token `[MASK]`.

There are other special tokens in BERT, such as `[CLS]` which represents the beginning of a sentence, `[SEP]` which represents the end of a sentence and the `[PAD]` token which is used when the maximum input length $M_{|i|}$ is greater than the sentence length $|s|$. For instance, given a $M_{|i|} = 5$ and the sentence "I like cake" the sentence representation would be "I like cake `[PAD]` `[PAD]`".

Additionally, BERT uses *WordPiece* [WSC+16] embeddings which break down words into sub-word units, also known as *wordpieces*. This token embeddings are summed with segment embeddings (which represent to which sentence the tokens belong) and position embeddings (which tell the relative position of a token in a sentence), resulting in the input representation.

These changes to the original *Transformer* architecture and the *fine-tuning* of *BERT* pre-trained *LM* on specific tasks resulted in *state of the art* results for the *GLUE* [WSM+18] benchmarks, which is a collection of diverse natural language understanding tasks [DCLT18].

## 2.4 Evaluation Metrics

Evaluation metrics are used to measure the performance of the model. There are many evaluation metrics to express different results and to apply on different problems, however we'll focus only on the metrics used on the [DAS+21].

When trying to evaluate a classification model it is essential to know how many samples were correctly and incorrectly classified, this information is often plotted in a *confusion matrix*, just like on 2.1.

| | Predicted 0 | Predicted 1 |
|---|---|---|
| Actual 0 | **TN** | **FP** |
| Actual 1 | **FN** | **TP** |

Table 2.1: Example binary confusion matrix.

One of the most common metrics is *precision*. It measures how many of the positive predictions made by the model are actually correct.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.17}$$

Even though it is a useful metric, when used by itself it leaves a lot to be said about the model's performance. The model can have a high precision and still miss a lot of positive cases, since the false negative samples are not considered with this metric.

False negatives are considered when calculating the model's *recall*, which measures how many positive predictions out all of the real positive instances are correct.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.18}$$

But, once again, this measure by itself tells very little. Imagine a binary classification problem where we have 70 positive samples and 30 negative samples and we opt for a *majority baseline*(classifying all the instance with the predominant class), the recall of the model would be equal to 1. This is obviously an extreme example but it shows the consequence of not choosing the right metrics.

If we think about the previous example and we opted for using precision we would have got a score of 0.7, exposing the model's inconsistency. By combining the 2 metrics we can have a better insight of the model's performance, and this is exactly what *f1-score* does, as it is an harmonic mean of precision and recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.19}$$

There are variants of *f1-score* that can be used in multiclass and multilabel problems such as *macro* and *micro*, which is the [DAS+21] official metrics for sub-task 1 and 3.

Macro f1-score computes the f1-score for each class and does an arithmetic mean. By doing so we're attributing the same weight to all classes which might not be true on imbalanced datasets.

$$\text{Macro F1-Score} = \frac{\sum_{i=1}^{n} \text{F1-Score}_i}{n} \tag{2.20}$$

When there's a clear class imbalance micro f1-score might be a best choice since it each instance equally which gives a better comprehension of the classifier overall performance.

$$\text{Micro F1-Score} = 2 \times \frac{\text{Micro-Precision} \cdot \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}} \tag{2.21}$$

Micro f1-score is calculated in the same way as the regular f1-score but it uses micro-average precision and recall.

## 2.5 Previous Approaches

### Multi-label Hate Speech and Abusive Language Detection in Indonesian Twitter

In 2019 a group of researchers published a paper [IB19] which discusses multi-label text classification for abusive language and hate speech detection including detecting the target, category, and level of hate speech in Indonesian Twitter using machine learning approach with Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest Decision Tree (RFDT) classifier and Binary Relevance (BR), Label Powerset (LP).

The problem only had 3 classes which makes it a good candidate for problem transformation. The researches did not only tried different models but also different ways to pre-process the text and feature extraction. In the end, the best results came out of word unigram (feature extraction), Random Forest using Label powerset as the transformation method with an accuracy of 77.36

On the following year, the same researchers release a new paper [ISB19] with some improvements, in it they proposed to also use part of speech (also known as grammatical tagging), emojis and the word2vec for word embeddings. They managed to get a minor upgrade to the previous accuracy getting 79.85%. However, this result was not obtained using word2vec, which had slightly worse results. This corpus used was relatively small and for that reason word2vec was not able to train the model properly.

### A multi-label ensemble predicting model to service recommendation from social media contents

The researcher's investigation [JPY22] goal was to design a predictive recommendations method that predicts the consumer recommendations in travel and tourism, particularly in the case of the airline. As a basic classification model to train the dataset *KNN*, *SVM*, *Multi-Layer Perceptron*, *Logistic Regression*, *Random Forests* and *Ensemble* methods were used. Further, the model was boosted by implementing partitioning methods, to partition the label space into lower spaces, such as *RAkELo* (Randomized K label set) and *Louvain*. This allowed to transform every label set into a multi-class classification problem.

The dataset includes categorical and text data but also numerical data, however, due to our problem we will only concern about the text data fields approach. In this paper, in order to make text processable *Word2Vec* was used.

The best results came from using an *MLP* with *Louvain* with a *Micro F1-score* of 0.9055 followed by the same model with *RAkELo*, with a score of 0.8987.

### Balancing Methods for Multi-label Text Classification with Long-Tailed Class Distribution

A common multilabel classification challenge is capturing label dependencies, it gets even more challenging when when class distribution is long-tailed. There are some methods to counter data imbalance such as *re-sampling* and *re-weight* however they are not very effective when strong labels dependency and class imbalance are present simultaneously as they usually result in oversampling favouring the predominant labels.

This paper's [HGK+21] authors introduce the application of balancing loss functions for multilabel text classification in order to tackle both problems simultaneously. Although *Binary Cross Entropy* (*BCE*) is commonly used for multilabel text classification this loss function is vulnerable to label imbalance, because of that the authors propose the usage of 3 different loss functions: *Focal Loss*, *Class-balanced focal loss* and *Distribution-balanced loss*. The idea is to reweight *BCE* so that rare instance labels get more attention.

To test the hypothesis 2 datasets were used: *Reuters-21578* and *PubMed*. While the first dataset had a reasonable number of labels (90) the latter had 18211 labels. They used *SVM*s with *Binary Relevance* and *TF-IDF* features as a baseline classifier. To test the loss functions they used a Bert pre-trained model. The evaluation metrics used were *Micro F1-score* and *Macro F1-score*.

For most of the *Reuters-21578* dataset the results were pretty similiar, however when it comes to classes with less than 8 samples the *BCE* presented a *Micro F1-score* score of 0.00. *Focal Loss* and *Class-balanced focal loss* presented similar results to the baseline, with a score of 23.08 *Distributed-balanced loss* on the other hand achieved a score of 48.89.

The improvements of using the proposed loss functions were clearer on the *PubMed* dataset, where they yielded scores ≈58 *Micro F1-score* score for the whole label space while *BCE* scored only 26.17.

### NLPIITR: RoBERTa Model with Data Augmentation for Persuasion Techniques Detection

On of the teams that took part in [DAS+21] is NLPIITR [GS21]. To address sub-task 1 they tried 13 different models to obtain results, with different pre-process techniques and ended up with 23 results.

It was found that strong results could be achieved by fine-tuning RoBERTa, however it was also found that some other models behaved fairly well when being part of a voting classifier, and requires way less computational resources than RoBERTa. Soft voting classifier and hard voting classifier were deployed to use traditional machine learning models for the task.

They tried multiples machine learning and deep learning approaches, such as Logistic Regression, Naive Bayes, SVMs, Random Forests, LSTMs, BERT and RoBERTa. Since some of the approaches support multi-class classification, they wrapped the these models in a OneVsRestClassifier (scikit-learn implementation of Binary Relevance).

To overcome the scarcity of data the researchers tried different data augmentation techniques. Some of them boosted the model's performance, however, others just confused the model, such as: random swap, synonym replacement, random deletion and random insertion. Back Translation, which is a classic data augmentation technique that translates the text data to another language and then it translates it back to the original language, was one of the techniques that boosted the model's performance. By applying it new text was generated while preserving the context of the sentence.

To pre-process the data the researchers created custom functions to remove special characters and punctuation and convert all the words to lower-case. After that they proceeded to remove the stopwords using the nltk corpus. They also experimented using Stemming and Lemmatization using nltk and realized that, even though it is computationally more complex, Lemmatization yielded better results.

In the end they were able to conclude that, on the dev-set, RoBERTa fine-tuned on the augmented data presented the best score with an Micro F1-Score of 0.509. The second best result was achieved with an ensemble of all machine learning models (13 models) with a hard-voting classifier of a with a threshold of 4 (which means the class is predicted if at least 4 classifiers classify it as positive). The Micro F1-Score was 0.440.

### YNU-HPCC: Combining ALBERT and Text-CNN for Persuasion Detection in Texts and Images

In order to tackle sub-task 1 [DAS+21] the team [ZWZ21] used a pre-trained ALBERT model and added a Text-CNN layer. By providing the hidden representation matrix outputted by ALBERT they were able to train the CNN layer with kernels of size 3, 4 and 5. The chose Adam as a model's optimizer and binary cross entropy as a loss function. The hyperparameters were obtained using grid-search.

Before committing to ALBERT they also experimented pre-trained BERT and RoBERTa but they failed to improve over ALBERT. Using the final configuration they managed to get a Micro f1-score of 0.472 on the dev set.

### AIMH: Multimodal Classification Using an Ensemble of Transformer Models

The AIMH team participated [MFGA21] on sub-task 1 and 3 [DAS+21], their main focus was sub-task 3 but they ended up using part of the model in order to participate on sub-task 1.

They started by proposing a baseline model which they called VTTE (Visual-Textual Transformer Encoder). The transformer encoder's visual and textual input features were extracted respectively from a CNN and a pre-trained BERT model. The textual and visual inputs were separated by the `[SEP]` special character embedding. The encoder output was followed by a fully connected layer and a sigmoid layer which would output the probabilities for each persuasion technique.

After establishing a baseline model they presented the DVTT (Double Visual Textual Transformer) which consists of 2 different transformers networks, able to process visual and textual inputs concurrently. The important aspect of DVTT is that each transformer is conditioned on the other modality , so that is possible for the whole architecture to jointly reason on the 2 modalities following 2 different paths. They do this by passing the sequence of textual embeddings to the visual transformer's encoder and the sequence of visual embeddings to the visual transformer's decoder, and vice-versa for the textual transformer. Each transformer is followed by a fully connected layer and and a sigmoid layer. The probabilities of each sigmoid layer are then averaged which gives the final probability for each label.

To tackle sub-task 1 they simply used the VTTE model, with a binary cross-entropy loss function, without the visual features which resulted in a Micro F1-Score of 0.539 on the test set. For sub-task 3, 6 models were trained. A ResNet50 pre-trained on image classification was used in all of them in order to encode the image set of features. The first model trained was the VTTE and the remaining models were variants of DVTT, with different parameters or different number of layers (encoder-decoder).

Overall the DVTT models had a better performance than the suggested baseline VVTE. For the final submission it was used an ensemble of the 6 models trained, where the probabilities were produced using a soft-voting classifier. This submission produced a Micro F1-score of 0.540 for the test set.

### MinD: Propaganda Detection using Transfer Learning and Multimodal Fusion

*MinD* took part on *SemEval-2021 Task 6* challenge, being the contest winners for *sub-task 1*. They published a paper [TGL+21] with their system's architecture regarding *sub-tasks* 1 & 3.

For *sub-task 1*, due to data's scarcity, transfer learning was used. Specifically by fine-tuning pre-trained large language models (*LLM*s) on an external dataset from *SemEval-2020 Task 11* [MBW+20] and then continuing to fine-tune then using the proper corpus [DAS+21]. An ensemble of transformers (*BERT*, *RoBERTa*, *XLNet*, *DeBERTa* and *ALBERT*), with *Focal Loss* (to address the imbalanced classes), was used. The model's probabilities were averaged and with that results, post-processing rules were applied. For instance, each bi-gram appearing more than 3 times is flagged as a repetition.

*Sub-task 3* is a multimodal task, combining text with images. The model trained for *sub-task 1* was used to address the textual elements. To perform feature extraction on images, they used *ResNet-34*, *OCR-based positional embeddings* and *Faster R-CNN*. The 2 modalities were combined using late fusion.

Just like expected, the predominant labels (*Name Calling* and *Loaded Language*) show a reasonable performance (0.667 and 0.819 Micro F1-score respectively). However, the ones with less than 20 training samples tend not to be predicted.

## 2.6 Summary

In this chapter we described some *State of the Art* concepts used in *multilabel* classification tasks concerning natural language processing and computer vision as well as some real world approaches on similar problems.

To do so we started by explaining some conventional *ML* algorithms that support *multilabel* classification (such as *KNN* and *Decision Trees*) and also went over some techniques that allow us to turn *multilabel* classification tasks into either *multiclass* or *binary* classification (such as *Label Powerset* and *Binary Relevance*).

Later, we talked about *DL* with more complex models like *LSTM*s and *Transformers* and how we can used pre-trained models as a leverage in our solution.

Following *ML* and *DL* we also presented some common *NLP* pre-processing techniques as well as *state of the art* words representation.

Evaluation metrics employed on this type of classification task were also discussed.

# 3

# Problem study

In this chapter we present the datasets used and do some exploratory work on them. Simple *ML* classifiers are used on the data in order to gain sensibility with the problem and the data.

## 3.1 Data analysis

The data we'll been working with has been provided by the organizing team of *SemEval's 2021 Task 6* [Sem]. To do so, English memes were collected from 26 *Facebook* groups regarding *politics*, *Covid-19*, *vaccines* and *gender equality* and gathering memes about such topics.

For this challenge the team considered a *meme* as a "photograph style image with a short text on top of it", excluding some of the data collected. For each meme, the textual content was annotated first and later the whole meme was annotated. Each of these annotation were performed in 2 phases: first the annotators independently annotated the memes; afterwards the annotators met with a consolidator do discuss and select the final gold labels.

The final annotated dataset consisted of 950 memes: 687 memes for training, 63 for development and 200 for training. While the maximum number of sentences in a meme is 13, the average number of sentences per meme is 1.68.

The data used for the 2 sub-tasks is the same, even though for sub-task 1 only text elements are used whilst sub-task 3 also use the original images. The dataset however differ on the $Y$ (predictions) axis due to the sub-tasks particularities. In order to give us a greater knowledge of the data the table 3.1 was provided, where we can understand the labels distribution for the 2 tasks.

| **Persuasion Techniques** | **Sub-task 1** | **Sub-task 3** |
|---|---|---|
| Loaded Language | 489 | 761 |
| Name Calling/Labeling | 300 | 347 |
| Smears | 264 | 602 |
| Doubt | 84 | 111 |
| Exaggeration/Minimisation | 78 | 100 |
| Slogans | 66 | 70 |
| Appeal to Fear/Prejudice | 57 | 91 |
| Whataboutism | 54 | 67 |
| Glittering Generalities (Virtue) | 44 | 112 |
| Flag-Waving | 38 | 55 |
| Repetition | 12 | 14 |
| Causal Oversimplification | 31 | 36 |
| Thought-Terminating Cliché | 27 | 27 |
| Black-and-White Fallacy/Dictatorship | 25 | 26 |
| Straw Man | 24 | 40 |
| Appeal to Authority | 22 | 35 |
| Reductio ad Hitlerum | 13 | 23 |
| Obfuscation, Intentional Vagueness, Confusion | 5 | 7 |
| Presenting Irrelevant Data (Red Herring) | 5 | 7 |
| Bandwagon | 5 | 5 |
| Transfer | — | 95 |
| Appeal to (Strong) Emotions | — | 90 |
| **Total** | **1642** | **2488** |

Table 3.1: Statistics about the persuasion techniques.

By looking at table 3.1, on the *Sub-task 1* column, we can spot a clear imbalance between classes, with *Loaded Language*, *Name Calling* and *Smears* appearing on over 25% of the samples while *Bandwagon*, *Irrelevant Data*, *Confusion*, *Reductio ad Hitlerum* and *Repetition* are present on less than 2% of the data.

The dataset was distributed in 2 different phases: **development** and **test**. During the first phase the training and development data were released, however no gold labels ($Y$) were released for the latter. During the test phase the labels for the *development* set were released as well as the test data, without the gold labels.

To understand if the datataset's splits (*train*, *dev/validation*, *test*) are balanced a graphic was plotted (3.1). When looking at it, we realize that not only the splits are not balanced, some classes are not even present in certain splits. For instance, *Black and white Fallacy*, *Red Herring* and *Obfuscation* are completely missing from the *development/validation split*.

The two datasets, as well as their partitions come in the format of *json* files. The content of those files is a list of dictionaries. Each dictionary represents a data entry and they consist of three fields: *id*, *labels*

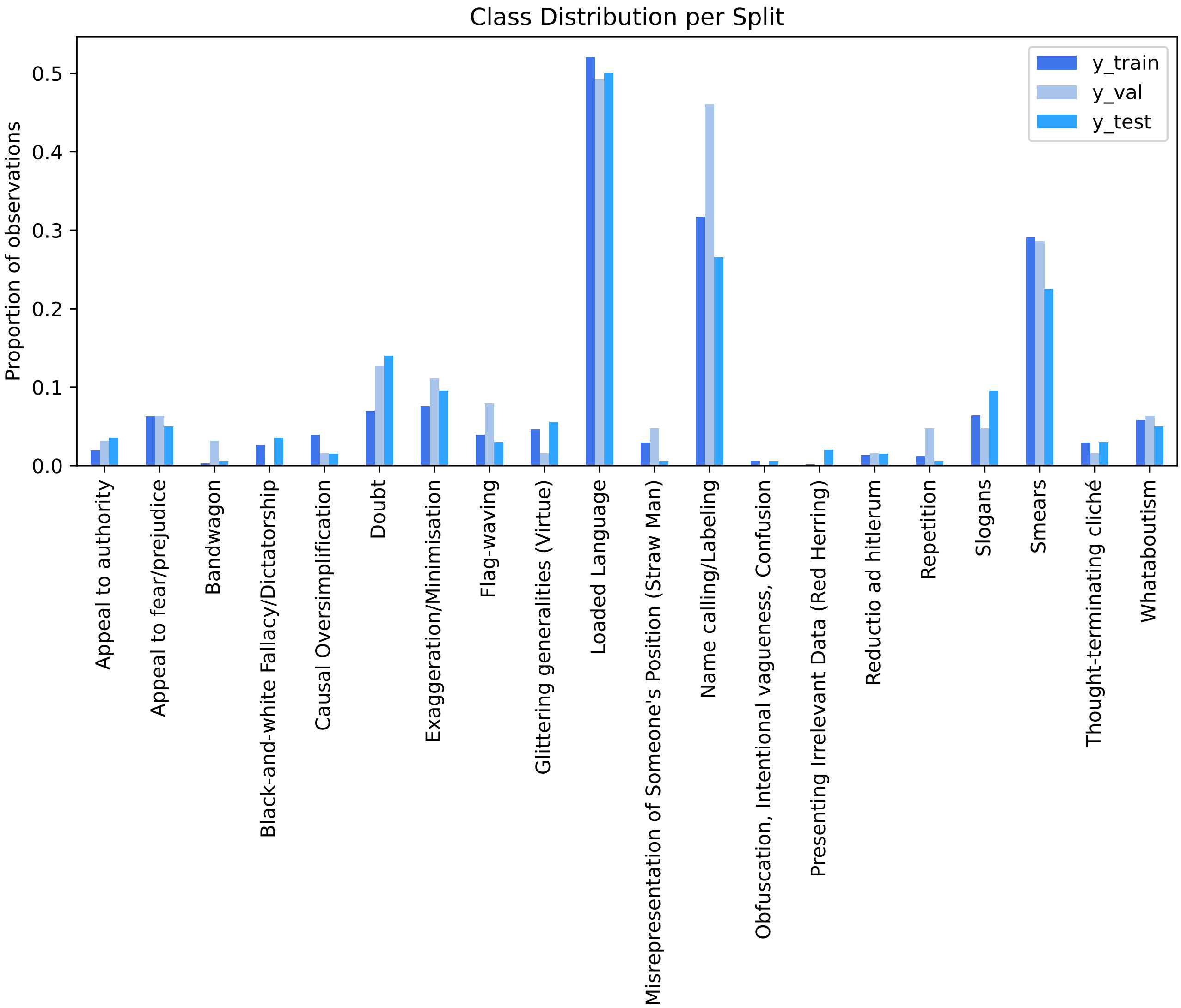


Figure 3.1: Class distribution per split on *sub-task 1*.

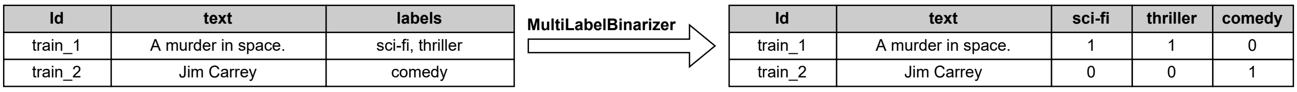
and *text*. The *labels* represents the predictions ($Y$) and is a list of strings for sub-taks 1 and 3. The *text* field is a string containing the textual elements of the original image. The *id* field is a string which maps each entry to an image, this field is particularly useful for sub-task 3.

## 3.2 Data pre-processing

### Labels

As told on this chapter the labels are given as a list of strings. This representation is not desirable because it makes it harder to pass to a dataframe and later to a classifier.

A proper way to represent the predictions would be to transform the list of categorical data into a list of binary values with *1* meaning the value is present and *0* the absence of it. *Sklearn* provides a function to do what I've just described, called *MultiLabelBinarizer*.



| Id | text | labels |
|---|---|---|
| train_1 | A murder in space. | sci-fi, thriller |
| train_2 | Jim Carrey | comedy |

| Id | text | sci-fi | thriller | comedy |
|---|---|---|---|---|
| train_1 | A murder in space. | 1 | 1 | 0 |
| train_2 | Jim Carrey | 0 | 0 | 1 |

Figure 3.2: Multilabel Binarizer transform.

The transformation of the data is represented on figure 3.2.

### Textual elements

In order to reduce the dataset's textual elements dimension and complexity a pre-processing pipeline was created. The pipeline has the following steps:

1. **Case converting**: The first thing to do is to lower case all the words on the sentence and remove the punctuation.
2. **Pos-Tagging**: Using *nltk* we perform a positional tagging to the sentence adding a syntactic meaning to each word. This function uses a pre-trained model to find the correct tags.
3. **Tokenization**: We opted for doing a word level tokenization.
4. **Stop-words and numbers removal**: The common words that in *nlp* problems do not add much meaning to the sentence are called stop-words. *Nltk* provides a list of english stop-words which we used to removed them from our corpus. The numbers were removed using a *regex* expression.
5. **Text Normalization**: Once again, *nltk* was used to lemmatize our tokens. Lemmatization consists of reducing a word to its root form, also known as *lemma*. Lemmatization can usually be improved when providing also syntactic meaning to a word, that's why *pos-tagging* was used. The lemmatizer chosen was the *WordNetLemmatizer*.

The results of pre-processing are presented on table 3.2. The reduction of the corpus dimension stems from the aggregation of words with the same lemma and stop-word removal. That way, only the most meaningful words will be converted to a vector during the feature extraction.

| | **Nº of Words** | **Nº of distinct words** |
|---|---|---|
| Unprocessed text | 16840 | 6427 |
| Pre-processed text | 9483 | 3092 |

Table 3.2: Corpus dimension before and after pre-processing

## 3.3 *ML* models

After cleaning the data we trained some models using classical *ML* approaches. These initial experiments should enable us to better understand the problems posed by the data. To perform the experiments *Python* libraries such as *Sklearn* provided the classifiers used and *Scikit-Multilearn* allowed us to use those classifiers with *PT* approaches.

According to the challenge's paper [DAS$^{+}$21] the *development* set could be used to train during the *test phase*, so we used it during the training process. To evaluate the models trained, *K-fold cross-validation* was used with $K = 5$.

Classifier's parameters where not chosen using advanced techniques but rather a random search of parameters. Multiple values were tested, however only the models yielding the best results are presented.

To convert the textual elements into features two approaches were considered. The first one uses a simple statistical method (*TF-IDF*) while the other tries a more sophisticated approach using neural networks to generate word embeddings (*Word2vec*).

*Sklearn* provides the *TfIdfVectorizer* class which was used. Some results of TF-IDF based models are shown in table 3.3.

| Model | Technique | Algorithm | N-Gram | Parameter | Param. Value | Micro F1 | Macro F1 | Time |
|---|---|---|---|---|---|---|---|---|
| M1 | - | Decsision Tree | (1,2) | class_weight | None | 0.306 | 0.133 | 0.68 |
| M2 | - | ExtraTrees | (1,2) | max_features | None | 0.317 | 0.126 | 0.70 |
| M3 | - | KNeighbors | (1,2) | n_neighbors | 20 | 0.203 | 0.047 | 0.18 |
| M4 | Binary Relevance | Decsision Tree | (1,1) | class_weight | None | 0.315 | 0.126 | 5.14 |
| M5 | Binary Relevance | ExtraTrees | (1,1) | max_features | None | 0.303 | 0.091 | 1.57 |
| M6 | Binary Relevance | KNeighbors | (1,2) | n_neighbors | 20 | 0.190 | 0.044 | 1.31 |
| M7 | Binary Relevance | Gauss. Naive Bayes | (1,2) | None | - | 0.302 | 0.086 | 2.55 |
| M8 | Label Powerset | Decision Tree | (1,1) | class_weight | None | 0.321 | 0.117 | 0.87 |
| M9 | Label Powerset | KNeighbors | (1,2) | n_neighbors | 20 | 0.120 | 0.029 | 0.10 |
| M10 | Label Powerset | Gauss. Naive Bayes | (1,2) | None | - | 0.349 | 0.085 | 1.76 |

Table 3.3: Results using TF-IDF as feature extraction technique.

| | F1-Score | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **M1** | **M2** | **M3** | **M4** | **M5** | **M6** | **M7** | **M8** | **M9** | **M10** |
| Appeal to Authority | - | ***0.222*** | - | 0.200 | - | - | 0.143 | - | - | - |
| Appeal to Fear/Prejudice | - | ***0.133*** | - | - | - | - | - | 0.111 | - | - |
| Bandwagon | - | - | - | - | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | ***0.400*** | 0.200 | - | 0.182 | - | - | - | - | - | - |
| Causal Oversimplification | - | - | - | ***0.182*** | - | - | - | - | - | - |
| Doubt | 0.053 | 0.093 | - | ***0.103*** | 0.063 | - | - | - | - | - |
| Exaggeration/Minimisation | 0.320 | 0.24 | - | 0.125 | 0.296 | - | 0.057 | ***0.353*** | 0.190 | 0.083 |
| Flag-Waving | ***0.400*** | 0.286 | 0.286 | 0.143 | 0.143 | 0.286 | - | 0.308 | - | - |
| Glittering Generalities (Virtue) | 0.154 | - | - | - | ***0.167*** | - | - | 0.154 | - | - |
| Loaded Language | 0.465 | 0.519 | 0.426 | 0.576 | 0.512 | 0.419 | ***0.673*** | 0.570 | 0.264 | 0.617 |
| Straw Man | - | - | - | - | - | - | - | - | - | - |
| Name Calling/Labeling | 0.304 | ***0.435*** | 0.133 | 0.385 | 0.341 | 0.069 | 0.351 | 0.287 | 0.125 | 0.364 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | - | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | 0.207 | 0.095 | 0.100 | 0.063 | - | 0.100 | 0.200 | 0.143 | - | ***0.231*** |
| Smears | ***0.366*** | 0.289 | - | 0.326 | 0.306 | - | 0.224 | 0.302 | - | 0.263 |
| Thought-Terminating Cliché | - | - | - | - | - | - | - | - | - | - |
| Whataboutism | - | - | - | ***0.231*** | - | - | 0.074 | 0.118 | - | 0.133 |

Table 3.4: F1-Score by class on each model using TF-IDF.

The best micro f1-score results came from using rule based models (decision trees) and *Label Powerset* (*LP*) with *Naive Bayes*. Some of these rule based models also presented the highest macro f1-score which indicates the models make consistent predictions across the label space and not just for the predominant labels. This can be observed on table 3.4. Still by looking at the table we can confirm that no classifier predicts any of the least representative labels (the ones that appear in $\leq 2\%$ of the samples).

In order to improve the previous model's results we tried to use *Word2vec* to extract information from the textual features . *Gensim* library provided a pre-trained *Word2vec* model, trained on the *Google News* dataset (containing around 100 billion words). The embeddings for each sentence were obtaining by using a simple mean of the sum of the word embeddings on said sentence.

| Model | Technique | Algorithm | Parameter | Param. Value | Micro F1 | Macro F1 | Time |
|---|---|---|---|---|---|---|---|
| MW1 | - | Decsision Tree | class_weight | None | 0.302 | 0.102 | 0.77 |
| MW2 | - | ExtraTrees | max_features | None | 0.298 | 0.111 | 0.20 |
| MW3 | - | KNeighbors | n_neighbors | 20 | 0.445 | 0.074 | 0.33 |
| MW4 | Binary Relevance | Decsision Tree | class_weight | None | 0.295 | 0.106 | 4.14 |
| MW5 | Binary Relevance | ExtraTrees | max_features | None | 0.293 | 0.108 | 0.11 |
| MW6 | Binary Relevance | KNeighbors | n_neighbors | 20 | 0.445 | 0.074 | 0.39 |
| MW7 | Binary Relevance | Gauss. Naive Bayes | None | - | 0.360 | 0.194 | 0.16 |
| MW8 | Label Powerset | Decision Tree | class_weight | None | 0.301 | 0.119 | 1.24 |
| MW9 | Label Powerset | KNeighbors | n_neighbors | 20 | 0.415 | 0.083 | 0.06 |
| MW10 | Label Powerset | Gauss. Naive Bayes | None | - | 0.400 | 0.091 | 0.10 |

Table 3.5: Results using Word2vec as feature extraction technique.

| | **F1-Score** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **MW1** | **MW2** | **MW3** | **MW4** | **MW5** | **MW6** | **MW7** | **MW8** | **MW9** | **MW10** |
| Appeal to Authority | ***0.143*** | - | - | - | - | - | ***0.143*** | - | - | - |
| Appeal to Fear/Prejudice | - | 0.114 | - | 0.154 | 0.061 | - | 0.151 | - | ***0.182*** | - |
| Bandwagon | - | - | - | - | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | - | - | - | - | - | - | ***0.200*** | - | - | - |
| Causal Oversimplification | - | 0.095 | - | 0.087 | - | - | 0.108 | ***0.333*** | - | - |
| Doubt | 0.140 | 0.150 | - | 0.100 | 0.179 | - | ***0.295*** | 0.098 | - | 0.286 |
| Exaggeration/Minimisation | 0.195 | 0.182 | - | 0.205 | 0.103 | - | ***0.329*** | 0.065 | - | 0.095 |
| Flag-Waving | 0.105 | 0.154 | - | 0.095 | - | - | ***0.200*** | ***0.200*** | - | - |
| Glittering Generalities (Virtue) | 0.074 | 0.167 | - | - | 0.182 | - | ***0.308*** | 0.083 | - | - |
| Loaded Language | 0.616 | 0.612 | ***0.727*** | 0.604 | 0.599 | ***0.727*** | 0.663 | 0.553 | 0.684 | 0.643 |
| Straw Man | - | - | - | - | - | - | ***0.069*** | - | - | - |
| Name Calling/Labeling | 0.352 | 0.254 | 0.306 | 0.313 | 0.301 | 0.306 | 0.425 | 0.366 | ***0.429*** | 0.323 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | ***0.286*** | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | 0.114 | 0.065 | - | 0.188 | ***0.267*** | - | 0.213 | - | - | - |
| Smears | 0.291 | 0.293 | 0.451 | 0.299 | 0.231 | 0.451 | ***0.489*** | 0.245 | 0.361 | 0.466 |
| Thought-Terminating Cliché | - | ***0.133*** | - | - | - | - | 0.130 | - | - | - |
| Whataboutism | - | - | - | 0.067 | ***0.207*** | - | 0.161 | 0.160 | - | - |

Table 3.6: F1-Score by class on each model using Word2Vec.

Using a more advanced techniques we were able to get better results for some of the models, specifically the ones using *KNeighbors* and *Naive Bayes*. These improvements were not seen on the models using decision trees. When using *TF-IDF* we capture the feature's importance whilst with *Word2vec* we capture the feature's meaning. The rules generated by the *decision tree* models tend to look at the feature importance, which might explain why there were no improvements.

The models with the best results were the ones using *KNeighbors* and *Naive Bayes*. Out of the four models, the one with the smaller *Micro F1-score* was the one using *Naive Bayes* with *BR*, nonetheless, this model also presents the highest *Macro F1-score*. This happens because the model is giving an higher importance to the least representative classes as it can be seen on table 3.6 where this model (*MW7*) outperforms most models for every class except for the most representative. To understand this behaviour we compared the table A.3 with A.4 and we infer that even though the model presents and high recall for most labels, the precision for those labels is low, which means the model is predicting a lot of false-positive cases.

When looking at the models created using simpler approaches we can say that *MW7* and *MW10* are at the same level as some of the models presented on [DAS+21].

Both models outperform the *Random Baseline* and TriHeadAttention's model in micro and macro F1-score. *MW10* also outperforms the micro F1-Score of the *Majority Baseline* (0.374) and *NLPIITR*'s model (0.379) however its macro F1-score is smaller than the latter (0.126). The *Majority Baseline* simply assumed that the predominant class occurred on every example.

Even though the model *MW7* does not outperform the *Majority Baseline* or *NLPIITR* in terms of micro F1-score it is worth noting it does outperform them in terms of macro F1-score.

## 3.4 Summary

Exploring the dataset revealed it has a small number of samples which is aggravated by how unbalanced the label distribution is, as 17 out of 20 labels appear in less than 10% of the samples. This makes the classification task for less represented labels a real challenge.

It was also explained how the dataset's pre-processing pipeline was created and its effect on the corpus dimension. The result of this pipeline was later fed into models using classical ML algorithms and PT techniques.

The ML models were created using TF-IDF and Word2Vec to extract the textual features. In general, trees presented better results when using TF-IDF and Naive Bayes performed better with Word2Vec. Even though some of these models produce interesting results they are not ideal which justify us to employ more sophisticated DL techniques such as Transformer based architectures, like the ones discussed on chapter 2.

# 4

# Proposed System

The studies conducted on chapter 3 revealed that traditional ML techniques by them-self do not provide desirable results for this problem which might have led the other participants on *SemEval* [DAS+21] to go with *State of the Art* techniques such as different *Transformer* architectures.

Their approaches vary between fine-tuning the transformers on the provided dataset, using an ensemble of transformers or using other networks on top of the fine-tuned transformer (such as LSTM and CNN).

Fine-tuning such architectures can be an expensive process, both in time and money. Our architecture's goal is not only to achieve the best results possible but also to try to reduce the computational cost of training our model. The last part however cannot be precisely computed as we don't know the machines in which other participants trained their models and how much time it took.

## 4.1 Architecture

Even though we cannot precisely calculate how expensive our architecture is when compared to others there are certain aspects that have been studied and we can take into account, such as the transformer's choice.

DistilBERT [SDCW19] has made significant improvements regarding these questions, as it reduces the original BERT model's size by 40% and its inference time is 60% faster while retaining 97% of its functionality. To perform knowledge distillation the DistilBERT model uses *Teacher-Student* architecture, where the *student* has a similar architecture to the original BERT model except for the *pooling* layer and it reduces the number of layers by a factor of 2.

A very straightforward approach would be to simply fine-tune a pre-trained DistilBERT model and adding a linear layer on top of it, and it would probably produce fair results, however there are other networks that can produce better results without much overhead. CNNs have proved to produce good results when it comes to multi-label text classification [Als21] [YCD$^+$20] as they are able to recognize patterns in the text. These patterns can be simpler or more complex depending on the kernel's size, for instance a kernel with size 2 would be similar to using bi-grams. When compared with LSTMs they are less expensive in time as they are highly parallelizable due to its non sequential nature.

Our model's architecture is based on these concepts with the addition of a couple of linear and dropout layers.

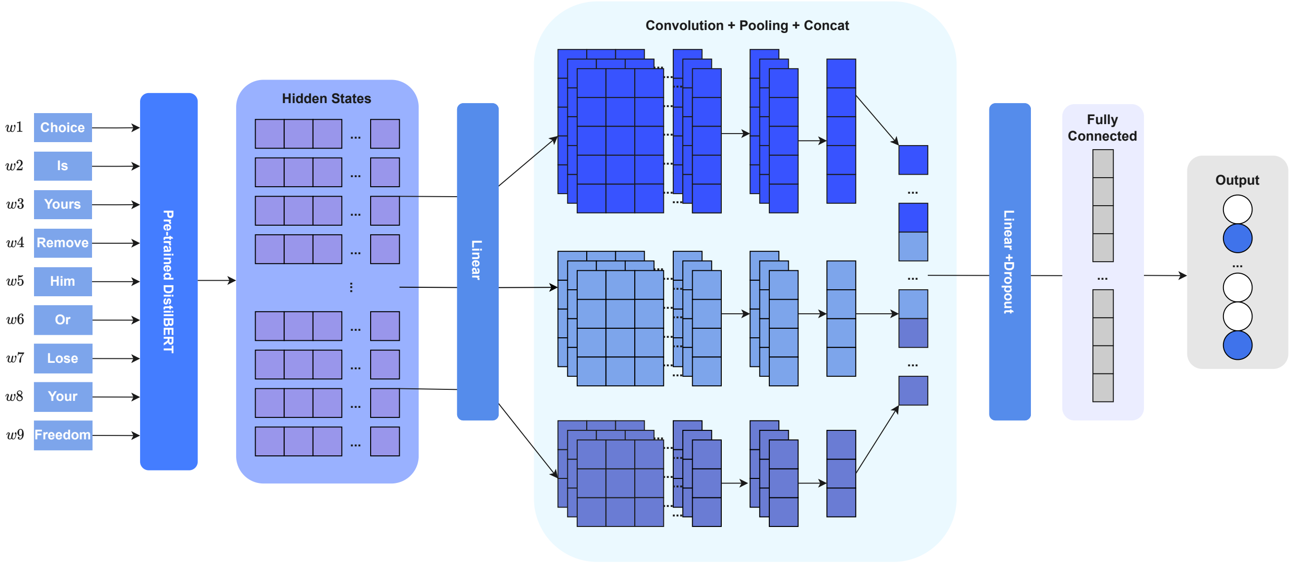


Figure 4.1: Final model architecture.

The text embeddings are passed into a pre-trained DistilBERT model which returns an input sequence representation, also known as *hidden state*, of size 768. The representation is passed to a linear layer followed and its output is then passed to a list of convolutional layers. A pooling operation is then applied to all the convolutional layers in order to extract the most important features. The result of the pooling operation of each layer is then concatenated in order to be passed to another group of linear and dropout layer which will then go into a fully connected layer with each neuron representing a label. The activation function used at the end of each linear layer (except for the last one) as well as after the convolutional layers is *elu*.

## 4.2 Implementation and deployment

The architecture described in the previous section was implemented in Python using the PyTorch [1] library. This library provides a vast set of common DL utilities with a robust documentation and a strong presence in the open-source community. The high usage of PyTorch amongst the open-source community stems from its pythonic way of writing code and the ease of usage when compared to other frameworks such as Tensorflow.

While most of the code was written using PyTorch some core ideas were implemented using the *Transformers* library from Hugging Face [2]. This library allowed us to load up to date pre-trained models with zero-effort instead of having us load pre-trained models from local files, amongst other things.

The model implementation using PyTorch is presented bellow:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from transformers import DistilBertModel

class ADPTModel(nn.Module):
  def __init__(self, num_classes, kernel_sizes, hidden_dim=768, num_filters=256, dropout_rate=0.05):
    super(ADPTModel, self).__init__()

    self.distilbert = DistilBertModel.from_pretrained('distilbert-base-uncased')

    self.l0 = nn.Linear(768, hidden_dim)

    self.convs = nn.ModuleList([nn.Conv1d(in_channels=hidden_dim, out_channels=num_filters, kernel_size=ks) for ks in kernel_sizes])

    self.l1 = nn.Linear(len(kernel_sizes) * num_filters, hidden_dim)
    self.d1 = nn.Dropout(dropout_rate)

    self.fc = nn.Linear(hidden_dim, num_classes)

  def forward(self, input_ids, attention_mask):
    x = self.distilbert(input_ids, attention_mask)
    hidden_state = x[0]
    hidden_state = F.elu(self.l0(hidden_state))

    hidden_state = hidden_state.permute(0, 2, 1) # (batch_size, embedding_dim, seq_len)

    x_convs = [F.elu(conv(hidden_state)) for conv in self.convs] # (kernel_len x (batch_size, num_filters, out_channels))
    x_pools = [F.max_pool1d(conv, conv.shape[2]).squeeze(2) for conv in x_convs] # (kernel_len x (batch_size, num_filters))
    x_cat = torch.cat(x_pools, dim=1) # (batch_size x (kernel_len*num_filters))

    out = F.elu(self.l1(x_cat))
    out = self.d1(out)
    out = self.fc(out)

    return out
```

The PyTorch implementation is very intuitive because we are able to define the model as a python class

[1] https://pytorch.org/

[2] https://huggingface.co/

which inherits the *nn* (neural networks) base module and simply demands us to define the layers used as well as the model's forward pass.

## 4.3 Experiment's methodology

After defining a base architecture there are many tweaks that can be made to the model, with that in mind it is important to define a methodology to ensure that we stay aligned with our goals.

Instead of using the *dev* dataset for validation, we will merge the *training* and *dev* datasets to create a new training set and we'll use use the stratified k cross-validation technique (with $K = 5$), provided by *scikit-multilearn*[3], just like we did on chapter 3. That way we can use the *test* dataset only to evaluate the final model which prevents us from overfitting it to this particular dataset.

There are four points worth exploring when it comes to the model's tweaking:

- **Loss function**: The loss function is a crucial aspect of the training process. Some loss functions can help us deal with the imbalanced class challenge.
- **Text pre-processing**: When using ML models text pre-processing was very important however, the transformer's embedding layer does some text pre-processing of it's own.
- **DistilBERT's fine-tuning**: When using a pre-trained model we can fine-tune the whole model on the new data, or simply fine-tune some specific layers (usually the last one).
- **Hyperparameters search**: We implemented our model in such a way that we can perform a search for a sub-optimal parameters such as kernel's size, number of convolutional layers, hidden layer's dimension, amongst others.

In the end it would also be useful to implement some basic models using other transformer architectures and compare its results with our final model as well as other participant's models.

## 4.4 Summary

In this chapter we went over our proposed system to tackle this problem.

We first explained our goal of achieving desirable results while reducing the computational expenses, followed by what we though was the best model's architecture to achieve it, using CNNs on top of smaller transformer architectures.

It was also explained which tools were chosen to implement our model as well as the reason of its choice, followed by the respective *PyTorch implementation*.

The final section consisted in the experiment's methodology where we explained how we would proceed to the tweaking and evaluation of the model and why we thought some factors such as the choice of the loss function or which layers of the transformer to tune are essential aspects of this research.

[3] http://scikit.ml/

# 5

# Results

To evaluate the proposed architecture we started with a baseline model using an arbitrary set of hyperparameters and followed the methodology defined on the the previous chapter to understand how different approaches impacted the model's performance.

All the experiments were performed on a Universidade de Évora's machine with the following specs:

- **CPU**: Intel Xeon Silver 4110 @ 2.10GHz;
- **GPUs**: 2x NVIDIA TITAN V 12GB;
- **RAM**: 95GB.

The models were trained inside a docker environment which allowed us to use the machine's hardware resources while not making any changes to the machine itself.

## 5.1 System evaluation

### 5.1.1 Baseline system

In order for our model to learn we need to choose a loss function and an optimizer. Instead of choosing a simpler optimizer such as Stochastic Gradient Descent (SGD) we opted for and adaptive optimizer such as *AdamW* mainly because it generally does provide a faster convergence without much tweaking and even though it has a computational overhead when compared to *SGD*, since most of the other participant's models also used this optimizer, its overhead is not taking into account when comparing with the remaining models.

For the loss function we went with BCE. Although this loss function's name might send us in the direction of a binary classification problem, it can also be used in multilabel classification problems by looking at the problem with binary relevance. Just like the optimizer, *Binary Cross Entropy* was also the loss function chosen in most models and is defined as it follows:

$$\text{BCE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \tag{5.1}$$

Where $y$ is the true value (0 if it does not belong to the class or 1 if it does) and $p$ the predicted probability of it belonging to the class. To simplify the notation we can define:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise.} \end{cases} \tag{5.2}$$

So that $\text{BCE}(p, y) = \text{BCE}(p_t) = -\log(p_t)$.

The set of hyperparameters for our baseline model is presented on table 5.1. The values chosen, except for kernel dimensions and hidden layer dimension, do not have any particular reason apart from being in a range of common values for each hyperparameter. Kernel dimension's value was the one chosen by [ZWZ21] and for the hidden layer dimension we went with the same dimension as the distilBERT hidden state.

| **Hyperparmeter** | **Value** |
|---|---|
| Learning rate | 3e-5 |
| Epochs | 10 |
| Batch size | 8 |
| Dropout rate | 0.2 |
| Number of filters | 128 |
| Kernel dimensions | [3,4,5] |
| Hidden layer dimension | 768 |

Table 5.1: Baseline system hyperparameters.

By running the experiments for the baseline model we confirmed the expected: a great improvement over the experiments on chapter 3 with a **micro f1-score of 0.516** yet there was still some inability to make correct predictions to the less represented classes, with 13 out of 20 classes having a f1-score of 0, as it can be observed on table 5.2. This inability is greatly reflected on the macro f1-score (0.116) which attributes the same weight to each class.

| **Technique** | **F1-Score** |
|---|---|
| Appeal to Authority | 0 |
| Appeal to Fear/Prejudice | 0.036 |
| Bandwagon | 0 |
| Black-and-White Fallacy/Dictatorship | 0 |
| Causal Oversimplification | 0 |
| Doubt | 0 |
| Exaggeration/Minimisation | 0.151 |
| Flag-Waving | 0.223 |
| Glittering Generalities (Virtue) | 0 |
| Loaded Language | 0.799 |
| Straw Man | 0 |
| Name Calling/Labeling | 0.535 |
| Obfuscation, Intentional Vagueness, Confusion | 0 |
| Presenting Irrelevant Data (Red Herring) | 0 |
| Reductio ad Hitlerum | 0 |
| Repetition | 0 |
| Slogans | 0.107 |
| Smears | 0.473 |
| Thought-Terminating Cliché | 0 |
| Whataboutism | 0 |

Table 5.2: F1-Score for each class on final architecture's baseline model.

### 5.1.2 Text pre-processing

The text fed to the baseline model did not go through any pre-processing pipeline, like we did for the ML models, it was raw text. This was possible because of the utilities provided by the *transformers*[1] library.

The library provides a base *Tokenizer* class with methods such as tokenize and encoding that combined can turn each token in a suitable numeric representation. Custom tokenizers like the one we used, *DistilBertTokenizer*, inherit the base class and re-implement some of these methods according to the architecture they're tied to.

Having a pre-processing pipeline implemented (described on chapter 3) we were able to investigate without extra effort the impact of pre-processing text from memes before feeding it to the transformer and if the benefits observed for the *ML* models can also be observed when using more robust models.

By looking at table 5.3 we can tell that even though the text pre-processing had impact on the model's performance, it was not positive, making the score for every class worse than the baseline model. What might have happened is that by pre-processing the text we lost important information that DistilBERT, unlike the ML models, is capable of handling an relies on.

To illustrate the previous statement, given the word "*walked*", the lemmatizer would have reduced the word to its lemma "*walk*". On the other hand if we passed the word to the DistilBERT tokenizer it would have splitted it into sub-words resulting in ["*walk*", "*##ed*"]. Not only does it preserve the meaning of the first sub-word, just like with the lemmatization, as it adds more information, in this case the verb tense.

[1] https://huggingface.co/docs/transformers/index

| Technique | F1-Score |
|---|---|
| Appeal to Authority | 0 |
| Appeal to Fear/Prejudice | 0 |
| Bandwagon | 0 |
| Black-and-White Fallacy/Dictatorship | 0 |
| Causal Oversimplification | 0 |
| Doubt | 0.036 |
| Exaggeration/Minimisation | 0 |
| Flag-Waving | 0.094 |
| Glittering Generalities (Virtue) | 0 |
| Loaded Language | 0.772 |
| Straw Man | 0 |
| Name Calling/Labeling | 0.550 |
| Obfuscation, Intentional Vagueness, Confusion | 0 |
| Presenting Irrelevant Data (Red Herring) | 0 |
| Reductio ad Hitlerum | 0 |
| Repetition | 0 |
| Slogans | 0.084 |
| Smears | 0.453 |
| Thought-Terminating Cliché | 0 |
| Whataboutism | 0 |

Table 5.3: F1-Score for each class on using text pre-processing on final architecture's baseline model.

### 5.1.3 Loss function

When dealing with imbalanced datasets in multilabel classification tasks the choice of the loss function can have an huge impact on the model's performance, specially on the least represented classes [HGK+21]. Although Binary Cross Entropy can be used it might not be the most suitable loss function for this problem as it does not deal with the imbalanced data problem. By not doing so, the system will struggle to properly compute the gradient for less popular classes which will result in the inability to accurately predict hard to classify examples.

Hard to classify examples are defined as misclassified ones (for instance a false positive). This issue was addressed in [LGG+17] where the researchers took Binary Cross Entropy as a starting point and added it a modulating factor of $(1 - p_t)^\gamma$. This type of loss would be called FL and would be given by:

$$\text{FL}(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{5.3}$$

Setting $\gamma > 0$ reduces the relative loss for well-classified examples putting more focus on hard misclassified examples. When an example is misclassified ($p_t \leq 0.5$), the modulating factor is nearer to 1 and the loss is unaffected. As $p_t$ gets closer to 1 the factor goes to 0 and the loss for well-classified examples is down-weighted.

In the original paper [LGG+17] the researchers found $\gamma = 2$ to produce the bests results, because of that we also set the parameter with the same value.

Although effective when it comes to classify hard to predict classes Focal Loss sets a trade-off: setting high $\gamma$, to sufficiently down-weight the contribution from easy negatives, may eliminate the gradients from

| | **F1-Score** | |
|---|---|---|
| **Technique** | **Focal Loss** | **Asymmetric Loss** |
| Appeal to Authority | 0.180 | 0.383 |
| Appeal to Fear/Prejudice | 0.210 | 0.299 |
| Bandwagon | 0 | 0 |
| Black-and-White Fallacy/Dictatorship | 0 | 0 |
| Causal Oversimplification | 0 | 0.069 |
| Doubt | 0.165 | 0.267 |
| Exaggeration/Minimisation | 0.212 | 0.281 |
| Flag-Waving | 0.205 | 0.385 |
| Glittering Generalities (Virtue) | 0.067 | 0.229 |
| Loaded Language | 0.794 | 0.806 |
| Straw Man | 0 | 0.109 |
| Name Calling/Labeling | 0.598 | 0.605 |
| Obfuscation, Intentional Vagueness, Confusion | 0 | 0 |
| Presenting Irrelevant Data (Red Herring) | 0 | 0 |
| Reductio ad Hitlerum | 0 | 0.100 |
| Repetition | 0 | 0.133 |
| Slogans | 0.227 | 0.300 |
| Smears | 0.456 | 0.507 |
| Thought-Terminating Cliché | 0 | 0.050 |
| Whataboutism | 0.114 | 0.231 |

Table 5.4: F1-score for each class using different loss functions.

the rare positive samples. This behaviour can be troublesome, specially in situations where predicting a false negative is more costly. To address this issue a new loss function was proposed: Asymmetric Loss (ASL) [BRZ$^+$20].

Asymmetric Loss relies on two principles, Asymmetric Focusing and Asymmetric Probability Shifting. The first principle decouples the focusing levels of the positive and negative samples so that $\gamma+$ and $\gamma-$ are the respective focusing parameters. Using it we can redefine the loss as:

$$\begin{cases} L+ = (1-p)^{\gamma+} \log(p) \\ L- = p^{\gamma-} \log(1-p) \end{cases} \tag{5.4}$$

This mechanism is able to reduce the contribution of negative samples to the loss when their probability is low, however it might not be very effective when class imbalance is very high. Rather than just reducing the contributions, Asymmetric Probability Shifting fully discards negative samples when their probability is very low. This principle, $p_m$, is given by:

$$p_m = \max(p - m, 0) \tag{5.5}$$

where the probability margin $m \geq 0$ is a tunable hyperparameter.

By integrating $p_m$ into $L-$ we get the definition of Asymmetric Loss:

$$ASL = \begin{cases} L+ = (1-p)^{\gamma+} \log(p) \\ L- = {p_m}^{\gamma-} \log(1-p_m) \end{cases} \tag{5.6}$$

ASL allows us to apply two types of asymmetry for reducing the contribution of easy negative samples to the loss function: soft thresholding via the focusing parameters $\gamma- > \gamma+$ , and hard thresholding via the probability margin $m$.

For our experiments we followed the values suggested by the original paper [BRZ+20] researchers with $\gamma+ = 0$, $\gamma- = 4$ and $m = 0.05$. By setting $\gamma+ = 0$ the positive samples will incur simple cross entropy loss, and control the level of asymmetric focusing via a single hyper-parameter, $\gamma-$.

The experiments made show the importance of choosing the right loss function as both Focal Loss and Asymmetric Loss improved our model's performance, specially when considering the less represented classes. This statement is confirmed when comparing tables 5.2 and 5.4.

| | **Micro F1-score** | **Macro F1-score** |
|---|---|---|
| Focal Loss | 0.523 | 0.162 |
| Asymmetric Loss | 0.525 | 0.238 |

Table 5.5: Micro and Macro F1-scores for different loss functions.

### 5.1.4 DistilBERT fine-tuning

Fine-Tuning a pre-trained transformer architecture can be an expensive process. In order to make the training faster transformer's layers can be frozen, which will reduce the number of tunable parameters preventing the weights from being updated. However, this approach can lead to performance loss.

To study the impact of freezing different layers we used as baseline our best model so far, the baseline architecture with asymmetric loss. To freeze the layers we started by freezing the initial layer and gradually freezing the layers that come after. The results for the experiments are detailed on table 5.6, where *F0* is the baseline model, *F1* freezes distilBERT's initial layer and *F6* freezes all distilBERT's layers.

| **Model** | **Micro F1-Score** | **Macro F1-Score** | **Parameters (Millions)** |
|---|---|---|---|
| F0 | 0.525 | 0.238 | 68 |
| F1 | 0.519 | 0.252 | 61 |
| F2 | 0.515 | 0.244 | 54 |
| F3 | 0.515 | 0.230 | 47 |
| F4 | 0.505 | 0.241 | 40 |
| F5 | 0.505 | 0.234 | 33 |
| F6 | 0.503 | 0.177 | 26 |

Table 5.6: Macro and Micro F1-Score freezing DistilBERT's layers.

The results show a slight decrease in micro f1-score after freezing the first 3 layers, however the macro f1-score does not decrease and when freezing only the first and second layers it improves. As we get closer to the final layers and freeze them the performance tends to deteriorate. The described behaviour is not surprising since the learnt parameters on the final layer are responsible for capturing complex patterns.

By looking at table 5.6 we observe that after freezing more than 3 layers the model's performance starts to decrease. Because of that we opted by freezing only 2 layers. Even though there's a slight decrease when it comes to the micro f1-score the macro f1-score increases and by freezing the 2 initial models we reduce the number of tunable parameters by 20%.

### 5.1.5 Hyperparameter search

An important part of training a model is to find suitable values for the model's parameters. Until now we have been using a set of values ( 5.1) naively chosen. In this sub-section we'll discuss different possible approaches to hyperparameter, which one did we chose and why.

Hyperparameter optimization is generally expressed as:

$$x^* = \arg\max_{x \in X} f(x) \tag{5.7}$$

Where $f(x)$ represents the return value of an objective function to minimize (for instance, the loss) or maximize (the score, for instance accuracy). $x^*$ is the set of hyperparameters that optimizes the return value, and $x$ can take on any value in the domain $X$.

The process described is the same one we use when evaluating a model. But doing this process manually can be very expensive, inefficient and boring. Rather than doing this manually there are simple techniques than can be used such as *Grid Search* and *Random Search*.

In Grid Search we define a set of values for each hyperparameter resulting in a grid which will be iteratively tested with all combinations of hyperparameters. With Random Search we start with the same principle but instead of trying all combinations we randomly sample values from the grid. The simplicity of both techniques comes at a cost: inefficiency. Grid Search is computationally inefficient, because it reuses bad values, Random Search overcomes this problem but does not guarantee the optimal solution within the grid.

Both approaches described previously do not take into account past experiments, which will lead to the re-usage of bad parameter's values. This limitation can be overcome by using Sequential model-based optimization (SMBO).The past experiment's results are used to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function, such as $P(\texttt{score}|\texttt{hyperparameters})$. This is achieved by creating a *surrogate* model, which is a substitute model that is easier to optimize for the objective function. The surrogate model does not try to mimic our original model but to understand the underlying patterns in data and which parameters influence the training. It is agnostic to our original model and tied to the SMBO approach in question. They are usually formulated as Random Forests, Mixture Density Networks or Multi Layer Perception, depending on the approach.

A popular SMBO approach is Tree Parzen Estimator (TPE) [BBBK11]. Unlike most bayesian SMBO methods, TPE formulates the surrogate model as $P(x|y)$ rather than $P(y|x)$. It is given as:

$$P(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{5.8}$$

Where $l(x)$ are the values for the bad objective functions, $g(x)$ are the values for the good objective functions and $y^*$ is a threshold. This means that, given an $y^* = 0,2$, when maximizing the objective function, a result is considered bad if it is smaller than the top 20% results of the previous experiments. The next step on TPE is how to evaluate an unobserved hyperparameter combination by calculating how promising it is, expressed by:

$$P = \frac{g(x)}{l(x)} = \frac{P(x|\text{good})}{P(x|\text{bad})} \tag{5.9}$$

Simply put, the promisingness of a combination of hyperparameters is given by the probability of the combination occurring given a good objective function over the probability of the combination ocurring given a bad objective function. The higher the score the more likely a combination of hyperparameter is to be a good fit.

There are many python libraries that provide implementations of TPE. We used *Tune*[2] due to the vast amount of hyperparameter search algorithms provided (because it wraps around other libraries such as *Optuna* and *HyperOpt*) and the introduction of schedulers (which allow early stop for bad performing trials). Being part of the *Ray*[3] ecosystem, *Tune* allows the experiments to make usage of multiple *GPU*s without much effort.

We thought of making use of the *ASHA* scheduler, however a scheduler might not be beneficial for our problem since we have a small set of data. *ASHA* is based on Successive Halving Algorithm (*SHA*) [JT15]. The way *SHA* works is by having a pool of global resources (number of iterations) which are splitted by the number of trials we want to perform, when the trials exhaust their resources a round is completed. After the round is completed the halving is performed, where the worst half of the models is discarded and the global pool of resources is restored. The process repeats itself until we have the best performing model. *ASHA* improves over *SHA* by being able to operate asynchronously and make dynamic resource allocation, rather than dividing them equally. However, since we ended up not using it we did not study these changes in detail.

Given our small dataset we decided that using a scheduler would add more overhead than benefits to the experiment, because of that we decided to not use any and simply let all the models train for the stipulated number of epochs. In our *Tune* configuration we simply specified TPE (via *HyperOpt*) as a search algorithm, the search space as described on table 5.7 and the number of trials we wanted to perform, which in this case were 10. For each trial we used stratified cross validation just like we did on previous experiments and we used the asymmetric loss function.

| **Hyperparameter** | **Type** | **Values** |
|---|---|---|
| Learning rate | Float | $[1e-6, 1e-4]$ |
| Dropout rate | Float | $[0.05, 0.45]$ |
| Number of filters | Int | $i \in [5, 8]$ for $2^i$ |
| Kernel dimensions | Choice | $[[1, 2], [3], [3, 4, 5]]$ |
| Hidden layer dimension | Int | $i \in [8, 11]$ for $2^i$ |

Table 5.7: Search space for hyperparameters.

For most parameters on the search space we specified values within a reasonable interval for the parameters. As for the kernel dimensions, we chose 3 different values, as it would not make any sense to choose a range. The first value came from the idea that n-grams $= (1, 2)$ could provide good results, as we've seen with ML techniques on chapter 3. The second value was chosen because we've seen kernel size $= 3$ being used on a couple of internet articles and academic papers [ZW17] [DGX18] and behaving fairly well. The last value was the value chosen for our baseline system, which has been explained before.

The experiments results are displayed on table 5.8 and show that the model with the highest micro f1-score is the model *H6*. Models *H10*, *H7* and *H5* also displayed good results, being just slightly worse than *H6*, besides that, model *H6* is slightly smaller than the other three in terms of trainable parameters, since it only uses a convolution kernel. *H6* has a small dropout rate however convolutional neural networks do not have tendency to overfit [PK17] and also since we have a small number of training samples, by setting an

[2] https://docs.ray.io/en/latest/tune/index.html

[3] https://www.ray.io/

| Model | Learning Rate | Dropout | Filters | Kernels | Hidden Layer Dim | Micro F1 | Macro F1 |
|---|---|---|---|---|---|---|---|
| H1 | 1.189e-04 | 0.175 | 128 | [3,4,5] | 256 | 0.528 | 0.201 |
| H2 | 6.790e-05 | 0.119 | 128 | [3] | 512 | 0.539 | 0.272 |
| H3 | 1.879e-05 | 0.159 | 64 | [3] | 1024 | 0.504 | 0.240 |
| H4 | 6.926e-05 | 0.103 | 64 | [1,2] | 512 | 0.536 | 0.246 |
| H5 | 3.353e-05 | 0.373 | 64 | [3,4,5] | 1024 | 0.547 | 0.223 |
| H6 | 4.813e-05 | 0.063 | 128 | [3] | 1024 | 0.556 | 0.246 |
| H7 | 3.993e-05 | 0.095 | 128 | [3,4,5] | 1024 | 0.551 | 0.233 |
| H8 | 6.925e-05 | 0.107 | 64 | [3,4,5] | 256 | 0.499 | 0.261 |
| H9 | 1.578e-05 | 0.305 | 128 | [3,4,5] | 256 | 0.519 | 0.231 |
| H10 | 2.433e-05 | 0.144 | 128 | [1,2] | 1024 | 0.555 | 0.212 |

Table 5.8: Models trained using TPE as search algorithm.

high dropout rate on the fully connected layer the network might forget important patterns learnt by the previous layers.

Until now we've been training all the models for 10 epochs, like *MinD* [TGL+21], however we thought it would be interesting to plot the macro and micro f1-score evolution as well as the loss throughout the epochs to see how the model behaves and if we should train it for more epochs in order to improve the results. We plotted the measures over the course of 15 epochs.

By looking at figure 5.1 we can see that the micro f1-score stops improving after only 3 epochs, having some minor fluctuations after it. When looking at the macro f1-score the story is different, the measure keeps improving until the 7th epoch which might tell that the model starts by predicting correctly only the most represented classes and it needs more training to understand the more complex patterns of the least represented classes.

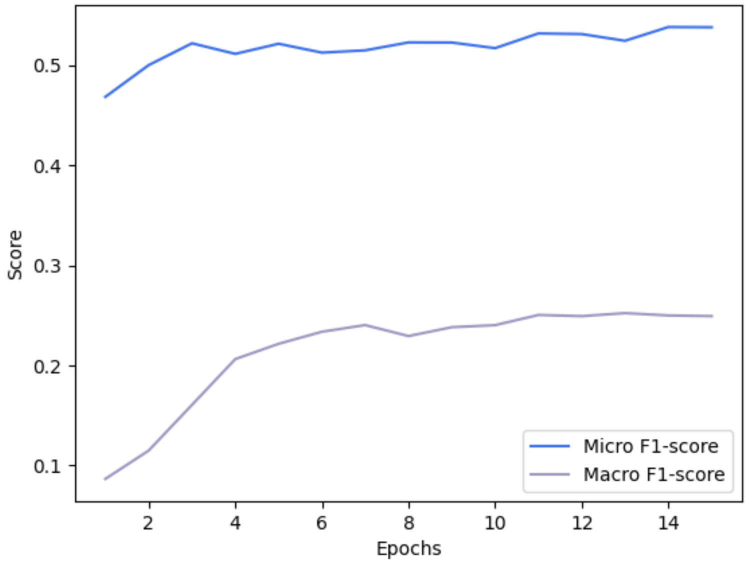


Figure 5.1: *H6* model score evolution.

The model's loss evolution surprised us because when we were doing the cross validation the micro f1-score was being consistent across folds, which led us to believe that the model was generalizing well however the plot (figure 5.2) tells otherwise. The validation loss starts by decreasing but after 3 epochs, around the same time in which the micro f1-score stops increasing, the loss stabilizes for 2 epochs and after the 6th epoch it starts continuously increasing which suggests the model is overfitting.

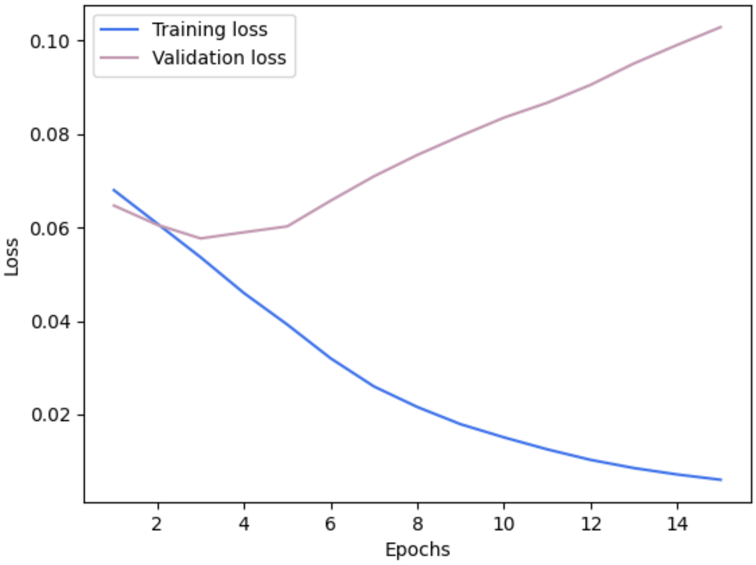


Figure 5.2: *H6* model loss evolution.

We tried to prevent this behaviour by adding a dropout layer at the end of the second linear layer and we were also counting on the *AdamW*'s L2 regularization technique through weight decay.

Afraid that we might have created a model too complex or used an improper loss function we took some steps back and made some experiments looking at the validation loss evolution. These experiments are explained on appendix B. From them, we concluded that the model is not overly complex and the loss function does not take a large part in the overfitting problem.

We were then left with two options, either to train model *H6* for fewer epochs with the configuration from table 5.8 or to tweak once more some of the parameters, this time, regarding the overfitting problem. Such parameters are the dropout rate, the weight decay on the optimizer and the learning rate.

By using a lower leaning rate ($4e-6$) the validation loss goes side by side with the training loss, which can show the model's convergence, however the obtained results were not the best specially for the least represented classes with a macro f1-score of 0.13. Evaluating the model's micro f1-score shows once more how imbalanced the data is as the score does not decrease that much (0.52).

## 5.2 Discussion

Looking at the prior experiments we can observe that text pre-processing does not have a positive impact on the proposed model's results, from figure 4.1, in fact it makes the results worse. Freezing some layers of the pre-trained distilbert can have small boost in terms of computational performance, as it reduces the number of tunable parameters, however there is a trade-off between micro f1-score and computational performance when freezing too many layers. For that reason it is fair to say that, although producing interesting results, they did not improve the model.

On the other hand, the loss function and hyperparameter choice had a massive impact on the model's results. With the proper choice we achieved acceptable results both in terms of micro and macro f1-scores using stratified cross-validation. The optimization however did not address the validation loss curve, as we were optimizing the model in terms of micro f1-score, which would give a better look of the model's overall performance. Looking at the validation loss curve is important as it can show signs of overfitting. We rectified this aspect by making some posterior experiences.

In the end, we were left with three versions of the same model configuration with small adjustments. Following the previous notation:

- **H6.1**: Original *H6* model trained for 4 epochs.
- **H6.2**: Original *H6* model trained for 15 epochs.
- **H6.3**: *H6* model with a learning rate ($4e{-}6$) trained for 15 epochs.

By evaluating the test set we were able to get the global results as well as class-wise and compare those results with *MinD*'s model, as they were the only ones in SemEval's task who published their f1-score for each class on the test set. The results are displayed on table 5.9

| | **F1-Score** | | | |
|---|---|---|---|---|
| **Technique** | **MinD** | **H6.1** | **H6.2** | **H6.3** |
| Appeal to Authority | 0 | 0.333 | 0.545 | 0 |
| Appeal to Fear/Prejudice | 0.522 | 0.333 | 0.333 | 0.278 |
| Bandwagon | 0 | 0 | 0 | 0 |
| Black-and-White Fallacy/Dictatorship | 0.400 | 0 | 0 | 0 |
| Causal Oversimplification | 0.500 | 0.286 | 0.222 | 0 |
| Doubt | 0.400 | 0.387 | 0.340 | 0.378 |
| Exaggeration/Minimisation | 0.550 | 0.375 | 0.542 | 0.333 |
| Flag-Waving | 0.615 | 0.286 | 0.444 | 0.316 |
| Glittering Generalities (Virtue) | 0.286 | 0.190 | 0.222 | 0.174 |
| Loaded Language | 0.819 | 0.813 | 0.823 | 0.805 |
| Straw Man | 0 | 0 | 0 | 0 |
| Name Calling/Labeling | 0.667 | 0.600 | 0.600 | 0.592 |
| Obfuscation, Intentional Vagueness, Confusion | 0 | 0 | 0 | 0 |
| Presenting Irrelevant Data (Red Herring) | 0 | 0 | 0 | 0 |
| Reductio ad Hitlerum | 0 | 0 | 0 | 0 |
| Repetition | 0 | 0 | 0 | 0 |
| Slogans | 0.154 | 0.303 | 0.250 | 0.242 |
| Smears | 0.511 | 0.468 | 0.486 | 0.467 |
| Thought-Terminating Cliché | 0 | 0 | 0 | 0 |
| Whataboutism | 0.375 | 0.190 | 0.333 | 0.292 |

Table 5.9: Comparing the final models results by class.

It is safe to say that both *H6.1* and *H6.3* models showed fairly good scores with the first showing slightly better scores across most classes. Model *H6.3* showed a clear difficulty, or even inability, to predict the 10 least represented classes. That can be caused by the small number of samples available for training, which did not allow the model to generalize that well. Assuming this is the cause, the fact that the model *H6.1* can predict some of these classes can potentially mean that either the model has a better generalization capability or it memorized part of the training set which enabled it to predict correctly some of the inputs for *Appeal to Authority* and *Causal Oversimplification*.

Model *H6.2* improved the scores for almost every class when compared with *H6.1* and showed very competitive results when compared with the *MinD* [TGL+21], the competition's best model. *H6.2* model's performance is somehow strange because even though its validation loss curve shows the model is overfitting the truth is that the results were very consistent across the training phase, using stratified cross-validation, and although we expected the model's performance to decrease substantially in the test set as it would be evaluating unseen data, the performance actually was very similar to the expected. This phenomenon can occur when the training and test data are very similar, which is a strong possibility because the data were extracted from multiple facebook groups related to politics and covid.

We recreated the SemEval's original paper [DAS+21] table regarding sub-task 1 and table 5.10 shows how our models would have behaved in a competition scenario. The table only contains submissions which are accompanied by a scientific paper.

| Rank | Model | Micro F1-Score | Macro F1-Score |
|---|---|---|---|
| 1 | MinD | 0.593 | 0.290 |
| 2 | Volta | 0.570 | 0.262 |
| 3 | **H6.2** | 0.551 | 0.257 |
| 5 | AIMH | 0.539 | 0.245 |
| 6 | **H6.1** | 0.526 | 0.228 |
| 7 | DistilBERT | 0.515 | 0.251 |
| 8 | LeCun | 0.512 | 0.227 |
| 11 | **H6.3** | 0.509 | 0.198 |
| 12 | NLyticsFKIE | 0.498 | 0.140 |
| 13 | RoBERTa | 0.497 | 0.240 |
| 16 | YNUHPCC | 0.493 | 0.263 |
| 19 | NLPIITR | 0.379 | 0.126 |

Table 5.10: SemEval's systems comparison.

*H6.2* performed worse than 2 other models however it should be taken in consideration that is also the smallest model out of all three. The MinD [TGL+21] model uses an ensemble of 5 transformers, such as RoBERTa and BERT and XLNET. Volta [GGM21] model is smaller when compared with MinD, since it uses only one transformer followed by two linear layers and dropout, the base transformer is however a RoBERTa Large, which has 355 million parameters.

While *MinD* model used focal loss in order to tackle class imbalance, *Volta* went with binary cross entropy. They did introduce however an weighting equation, made specifically for this problem, which shows that even though asymmetric loss granted us some good results out of the box, it is also possible to achieve good results with binary cross entropy and some tweaking.

The training configuration for both these models left us wondering if they do not also suffer from some kind of overfitting as both of them are trained with a learning rate of $(2e{-}5)$ and $(1e{-}5)$ respectively. *MinD* also trained their model for 10 epochs and used the same optimizer with the default weight decay (L2 regularization) however since they were able to use augmented data that might have stopped the model from overfitting, but there's no evidence about that. For the *Volta* model it seems even harder to believe that the model does not overfit because not only they did not use any additional data but also didn't use any regularization besides a dropout layer with a 0.1 dropout rate. Once again we cannot affirm that overfit occurs in the model as we do not know the number of epochs used to train that model nor about its eventual overtfitting.

*LeCun*'s [AQA21] model was also an ensemble of transformers containing one DeBERTa Large and four RoBERTa Large. They used different sequence max length, batch size, learning rate and number of epochs for each part of the ensemble. Each ensemble's component was trained for either two, three or four epochs, probably because they also spotted some overfitting after that number of epochs. In the end, they used a soft voting classifier with a 0.3 threshold for each label. *H6.1* was able to reach a better score on the test set showing that altough *LeCun*'s solution is very interesting it can be a bit of overkill and similar results can be obtained with less computational exhaustive systems.

The *YNUHPCC* [ZWZ21] model is the closer to ours from an architectural point of view, as we have drawn inspiration from it. However, instead of using DistilBERT they used ALBERT and binary cross entropy instead of using of asymmetric loss. They are the only team to mention worries regarding overfitting, specially when fine-tuning more robust models such as BERT or RoBERTa which they tried before going with ALBERT. Because of that not only did they used a lighter model, with less tunable parameters, but also used a smaller learning rate ($5\mathrm{e}{-6}$) like we did with *H6.3*. Despite getting an higher micro f1-score with *H6.3*, *YNUHPCC* got an higher macro f1-score using binary cross entropy. On their paper they do not mention any weighting function or any tweak to the base formula of BCE. We are led to believe that the increased macro f1-score was due to the number of epochs their model was trained for (300). *YNUHPCC* model also used Batch Normalization in-between linear layers. We also tried using it in our model but we did not observe any gains either in performance or validation loss which is not surprising given that this normalization method is seldom used in NLP problems, with Layer Normalization being more common [SYG+20]. The latter was also used but did not show any significant improvements.

At the end of our experiments we trained two additional models with the following architecture: a transformer, followed by a linear layer, a dropout layer and a fully connected linear layer. One of the models used DistilBERT while the other used RoBERTa. This can be seen as plain fine-tuning the previous transformers. We trained both of them for 4 epochs. Both these models, ranked 7th and 13th respectively, had a worst performance on the test set that the *H6.1* model. The simple DistilBERT model was able to predict two labels that none of our models were: *Black-and-White Fallacy* and *Straw Man*. Which means that although we passed the hidden states learnt by the distilbert model to a convolutional layer in order to extract local patterns on top of the global relationships this might have confused the model, preventing it to recognizing certain classes. It is also worth mentioning that despite getting similar micro f1-scores the distilBERT model processed an epoch 5 times faster than RoBERTa (4s vs 21s), making the first a suitable transformer for this task.

# 6

# Conclusion

The final chapter of this work contains a reflection of all the exploratory work we did regarding this problem, what we were able to accomplish, what we went wrong and why it might have happened.

## 6.1 Global overview

In order to achieve this work's goal, the automatic detection of persuasion attempts on social networks (proposed in a SemEval challenge), we employed natural language processing, machine learning and deep learning techniques. The given dataset for this problem had a small dimension with only 950 samples from which 200 were used for evaluation, leaving us with 750 samples for training which is a relatively small number when there are 20 possible persuasion techniques. Besides the dataset's small dimension the task is even more challenging due to the class imbalance with 12 classes appearing in less than 5% of the samples and 3 classes appearing on more than 25% of the samples, when looking only at the textual elements.

Given the small number of training samples we combined ML and NLP techniques to create a simple model capable of solving the targeted problem. The usage of techniques such as stop-word removal and lemmatization helped us to reduce in half the corpus's number of words which is very helpful when using ML techniques on text. We choose Decision Trees, Extra Trees, Gaussian Naive Bayes and K-Nearest Neighbors as our base classifiers. Since this problem is multi-label classification problem and some of these algorithms do not support it, in sklearn, we wrapped them in problem transformation techniques like Binary Relevance and Label Powerset. In order to feed the textual elements to the classifiers we experimented with two methods: TF-IDF and Word2Vec. The first method provided better results for tree-based classifiers but the latter outperformed it when it comes to the remaining classifiers. The best results came from using Gaussian Naive Bayes with Binary Relevance and Word2Vec embeddings with a micro f1-score of 0.360 and macro f1-score of 0.194.

Although ML techniques helped us set a valid baseline, when compared to SemEval's challenge best submissions they were lacking, which justify the usage of more sophisticated DL techniques. We opted for using a relatively small pre-trained transformer, DistilBERT, so that we could make use of Transfer Learning in order to mitigate the scarcity of data. This transformer was followed by linear and convolutional networks.

Our proposed model was iterated with different experiments where we found that pre-processing the textual elements did not improve the model, making its performance in fact worse. We also found out that for this problem, when it comes to the trade-off between computational expenses and the model's performance the best thing to do is to freeze the first two distilBERT's layers and fine-tune only the remaining. However, the experiment that gave us the highest gains was the choice of the loss function when training the model. Asymmetric Loss showed us the best results as, by definition, it tackles the class imbalance problem in multi-label classification problems. At last we used TPE method to perform hyperparameter search for parameters such as the number and size of convolutional kernels, the number of filter, the learning rate, dropout rate and the hidden layers dimension. When investigation the loss curve for the validation set we observed some signs of overfitting which led us to investigate it and try less complex architectures, train our model for less number of epochs or reduce the learning rate. We discovered that reducing the model complexity by itself did not solve the problem however changing the number of epochs or the learning rate (while maintaining the original architecture) could help.

We were left with three models: one without any signs of overfitting, one with minor signs and one with a very steep validation loss curve, with 0.509, 0.526 and 0.551 micro f1-score respectively. Strangely the last model showed similar performance both on the validation set and the test set. Although none of our models was able to outperform the best submission, the last two were able to compete with the top-5 submissions while using a model with a much smaller number of tunable parameters.

By the end of this work we partially achieved what we had initially proposed. We were able to create a usable model, with its due limitations, to detect persuasion attempts on social networks given textual elements. Unfortunately, we were not able to tackle the multimodal challenge, using text and images, which would have been very interesting as text when combined with the images can drastically have a different intent. The sub-task 1 required a deeper study that what we had initially though which eventually took more time than estimated, leaving the sub-task 3 out of scope for this thesis. Although we did not implement a solution for sub-task 3 we studied what the other participants had done, which is documented on chapter 2, providing us important information for future work regarding the multimodal classifier.

We also learnt how difficult it is to build an entirely accurate robust global system to identify persuasion attempts due to various factors such as the wide array of topics, the need to keep the system information up to date and the difficulty to relate different information. Another obstacle for this kind of task is the vocabulary used on social media where we can find a lot of slang and words with typos. Although text pre-processing can help regarding this problem this is not always the case. Researchers [Moh18] suggest that sometimes text pre-processing, if not made with maximum caution, can deteriorate the model's performance.

## 6.2 Future Work

Although we are happy with the results we have achieved there is still a lot of room for improvement. One of the biggest challenges we've had was the scarcity of the data. To overcome it we can try to use similar data from a previous SemEval challenge [MBW+20]. Despite having a slightly different scope, the previous challenge also had most of the techniques we are trying to predict. We think this can be a valid way to augment our dataset without sacrificing the quality of data, unlike translation and word replacement techniques where changing a word might change the sentence meaning, producing misleading data.

Given the class imbalance it would also be interesting to experiment an ensemble of classifiers, with a global classifier, like the one we had, and a couple of simpler classifiers that would only predict two or three of the least represented classes. In the end we could average the probabilities for each class. By having those multiple classifiers targeting specific classes the model could explicitly focus on those, possibly improving the overall performance while adding some computational overhead.

Leaving the multimodal challenge out of the thesis was a decision brought by the time restraints we faced during this work, nonetheless we would like to tackle this challenge by experimenting both early fusion and late fusion techniques. Early fusion technique extracts the features from both textual and visual elements and combines them into a unique representation, by doing so the classifier captures the underlying relationship between the two modalities. Late Fusion, on the other hand, represents each modality independently and and only combines the final result for each modality at the end with approaches such as averaging and weighting. While it can provide fairly good results, as we've seen for *MinD* [TGL+21], late fusion is a sub-optimal approach since two modalities on its own might not contain any persuasion technique or hateful discourse but when combined get a new intent, just like it has been shown on [KFM+21]. The main reason for us to consider its adoption is the fact that it is a computationally less expensive approach and it allows us to reuse our textual model. With early fusion we would not be able to reuse our model from sub-task 1 and would have to think in a new model from the ground, fortunately there are frameworks such as Facebook's *MMF* [1] which provide us with state of the art visual and textual models allowing us to create multimodal models writing less boilerplate code.

Ultimately, the work we have produced so far can be useful for automatic content moderation in any platform as we plan on making our models available online for free use. It might not have the robustness of a *Meta* or *Twitter* powered models but smaller platforms, who don't want to put a lot of effort into this topic, could really benefit of this by using it to enforce some of their community guidelines without having a human person reviewing all the reported content.

[1] https://mmf.sh/

# A

# ML Models

This appendix contains the *Precision* and *Recall* tables for each class for the models created using conventional Machine Learning techniques.

The tables are complementary to the ones on chapter 3 where the tables of *F1-Score*, which is an harmonic mean of precision and recall, are presented.

By having these tables in here we can have a deeper understanding of the results for each class and infer new insights such as if a class is predicting a lot of false-positives.

In the following pages, the first two tables are related to the models which used TF-IDF while the last two relate to the ones using Word2Vec.

| | **Precision** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **M1** | **M2** | **M3** | **M4** | **M5** | **M6** | **M7** | **M8** | **M9** | **M10** |
| Appeal to Authority | - | ***0.500*** | - | 0.333 | - | - | 0.143 | - | - | - |
| Appeal to Fear/Prejudice | - | ***0.200*** | - | - | - | - | - | 0.125 | - | - |
| Bandwagon | - | - | - | - | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | ***0.667*** | 0.333 | - | 0.250 | - | - | - | - | - | - |
| Causal Oversimplification | - | - | - | ***0.125*** | - | - | - | - | - | - |
| Doubt | 0.100 | 0.133 | - | 0.182 | ***0.250*** | - | - | - | - | - |
| Exaggeration/Minimisation | 0.667 | 0.500 | - | 0.154 | 0.500 | - | 0.063 | 0.400 | ***1.0*** | 0.200 |
| Flag-Waving | 0.500 | ***1.00*** | ***1.00*** | 0.125 | 0.125 | ***1.0*** | - | 0.286 | - | - |
| Glittering Generalities (Virtue) | 0.500 | - | - | - | ***1.0*** | - | - | 0.500 | - | - |
| Loaded Language | 0.574 | ***0.707*** | 0.522 | 0.662 | 0.611 | 0.522 | 0.648 | 0.646 | 0.586 | 0.579 |
| Straw Man | - | - | - | - | - | - | - | - | - | - |
| Name Calling/Labeling | 0.360 | 0.513 | ***0.571*** | 0.375 | 0.429 | 0.400 | 0.274 | 0.318 | 0.364 | 0.324 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | - | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | 0.300 | 0.500 | 1.0 | 0.077 | - | ***1.0*** | 0.273 | 0.222 | - | 0.429 |
| Smears | ***0.405*** | 0.355 | - | 0.319 | 0.325 | - | 0.169 | 0.317 | - | 0.241 |
| Thought-Terminating Cliché | - | - | - | - | - | - | - | - | - | - |
| Whataboutism | - | - | - | 0.188 | - | - | 0.059 | 0.143 | - | ***0.200*** |

Table A.1: Precision by class on each model using TF-IDF.

| | **Recall** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **M1** | **M2** | **M3** | **M4** | **M5** | **M6** | **M7** | **M8** | **M9** | **M10** |
| Appeal to Authority | - | ***0.143*** | - | ***0.143*** | - | - | ***0.143*** | - | - | - |
| Appeal to Fear/Prejudice | - | ***0.100*** | - | - | - | - | - | ***0.100*** | - | - |
| Bandwagon | - | - | - | - | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | ***0.286*** | 0.143 | - | 0.143 | - | - | - | - | - | - |
| Causal Oversimplification | - | - | - | ***0.333*** | - | - | - | - | - | - |
| Doubt | 0.036 | ***0.071*** | - | ***0.071*** | 0.036 | - | - | - | - | - |
| Exaggeration/Minimisation | 0.211 | 0.156 | - | 0.105 | 0.211 | - | 0.053 | ***0.316*** | 0.105 | 0.0053 |
| Flag-Waving | ***0.333*** | 0.167 | 0.167 | 0.167 | 0.167 | 0.167 | - | ***0.333*** | - | - |
| Glittering Generalities (Virtue) | ***0.091*** | - | - | - | ***0.091*** | - | - | ***0.091*** | - | - |
| Loaded Language | 0.390 | 0.410 | 0.360 | 0.510 | 0.440 | 0.350 | ***0.700*** | 0.510 | 0.170 | 0.660 |
| Straw Man | - | - | - | - | - | - | - | - | - | - |
| Name Calling/Labeling | 0.264 | 0.378 | 0.075 | 0.396 | 0.283 | 0.038 | ***0.491*** | 0.264 | 0.075 | 0.451 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | - | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | ***0.158*** | 0.053 | 0.053 | 0.053 | - | 0.053 | ***0.158*** | 0.105 | - | ***0.158*** |
| Smears | ***0.333*** | 0.244 | - | ***0.333*** | 0.289 | - | ***0.333*** | 0.289 | - | 0.289 |
| Thought-Terminating Cliché | - | - | - | - | - | - | - | - | - | - |
| Whataboutism | - | - | - | ***0.300*** | - | - | 0.100 | 0.100 | - | 0.100 |

Table A.2: Recall by class on each model using TF-IDF.

| | Precision | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **MW1** | **MW2** | **MW3** | **MW4** | **MW5** | **MW6** | **MW7** | **MW8** | **MW9** | **MW10** |
| Appeal to Authority | ***0.143*** | - | - | - | - | - | 0.095 | - | - | - |
| Appeal to Fear/Prejudice | - | 0.080 | - | 0.125 | 0.043 | - | 0.093 | - | ***1.0*** | - |
| Bandwagon | - | - | - | | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | - | - | - | - | - | - | ***0.333*** | - | - | - |
| Causal Oversimplification | - | 0.056 | - | 0.050 | - | - | 0.059 | ***0.333*** | - | - |
| Doubt | 0.200 | 0.250 | - | 0.094 | 0.179 | - | 0.217 | 0.154 | - | ***0.429*** |
| Exaggeration/Minimisation | 0.182 | 0.214 | - | 0.200 | 0.100 | - | 0.217 | 0.083 | - | ***0.500*** |
| Flag-Waving | 0.077 | ***0.143*** | - | 0.067 | - | - | 0.125 | 0.250 | - | - |
| Glittering Generalities (Virtue) | 0.063 | 0.154 | - | - | 0.182 | - | ***0.195*** | 0.077 | - | - |
| Loaded Language | 0.622 | 0.594 | 0.596 | 0.571 | 0.574 | 0.596 | ***0.657*** | 0.538 | 0.603 | 0.656 |
| Straw Man | - | - | - | - | - | - | ***0.036*** | - | - | - |
| Name Calling/Labeling | 0.306 | 0.231 | ***0.406*** | 0.290 | 0.250 | ***0.406*** | 0.333 | 0.308 | 0.344 | 0.375 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | ***0.250*** | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | 0.125 | 0.083 | - | ***0.231*** | 0.363 | - | 0.179 | - | - | - |
| Smears | 0.246 | 0.239 | 0.341 | 0.258 | 0.203 | 0.341 | 0.362 | 0.213 | 0.273 | ***0.414*** |
| Thought-Terminating Cliché | - | ***0.111*** | - | - | - | - | 0.075 | - | - | - |
| Whataboutism | - | - | - | 0.050 | ***0.158*** | - | 0.096 | 0.133 | - | - |

Table A.3: Precision by class on each model using Word2Vec.

| | Recall | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **MW1** | **MW2** | **MW3** | **MW4** | **MW5** | **MW6** | **MW7** | **MW8** | **MW9** | **MW10** |
| Appeal to Authority | 0.143 | - | - | - | - | - | ***0.286*** | - | - | - |
| Appeal to Fear/Prejudice | - | 0.200 | - | 0.200 | 0.100 | - | ***0.400*** | - | 0.1 | - |
| Bandwagon | - | - | - | | - | - | - | - | - | - |
| Black-and-White Fallacy/Dictatorship | - | - | - | - | - | - | ***0.143*** | - | - | - |
| Causal Oversimplification | - | 0.333 | - | 0.333 | - | - | ***0.667*** | 0.333 | - | - |
| Doubt | 0.107 | 0.107 | - | 0.107 | 0.179 | - | ***0.464*** | 0.071 | - | 0.214 |
| Exaggeration/Minimisation | 0.211 | 0.158 | - | 0.211 | 0.105 | - | ***0.684*** | 0.053 | - | 0.053 |
| Flag-Waving | 0.167 | 0.167 | - | 0.167 | - | - | ***0.500*** | 0.167 | - | - |
| Glittering Generalities (Virtue) | 0.091 | 0.182 | - | - | 0.182 | - | ***0.727*** | 0.091 | - | - |
| Loaded Language | 0.610 | 0.630 | ***0.930*** | 0.640 | 0.620 | ***0.930*** | 0.670 | 0.570 | 0.790 | 0.630 |
| Straw Man | - | - | - | - | - | - | ***1.0*** | - | - | - |
| Name Calling/Labeling | 0.415 | 0.283 | 0.245 | 0.340 | 0.377 | 0.245 | ***0.585*** | 0.453 | 0.566 | 0.283 |
| Obfuscation, Intentional Vagueness, Confusion | - | - | - | - | - | - | - | - | - | - |
| Presenting Irrelevant Data (Red Herring) | - | - | - | - | - | - | - | - | - | - |
| Reductio ad Hitlerum | - | - | - | - | - | - | - | ***0.333*** | - | - |
| Repetition | - | - | - | - | - | - | - | - | - | - |
| Slogans | 0.105 | 0.053 | - | 0.158 | 0.2111 | - | ***0.263*** | - | - | - |
| Smears | 0.356 | 0.378 | 0.667 | 0.356 | 0.267 | 0.667 | ***0.756*** | 0.289 | 0.533 | 0.533 |
| Thought-Terminating Cliché | - | 0.167 | - | - | - | - | ***0.500*** | - | - | - |
| Whataboutism | - | - | - | 0.100 | 0.300 | - | ***0.500*** | 0.200 | - | - |

Table A.4: Recall by class on each model using Word2Vec.

# B

# Taking a step back

After realizing the tweaked model following the proposed architecture could be overfitting, by looking at figure 5.2, we thought about taking a step backwards and inspect if we had made any mistakes during our research process.

## B.1 Looking at the different loss functions

The change that required less effort was to look at the model using the loss functions mentioned previously and inspect how the validation loss would evolve.

When comparing figures 5.2 and B.1 we observe a smoother curve for validation loss on the latter, however the model using FL still show signs of overfitting around the same number of epochs. When comparing the score evolution, figures 5.1 and B.2, of both models we see that around the time they start showing signs of overfitting both models display similar micro f1-scores, however when looking at the macro f1-score we

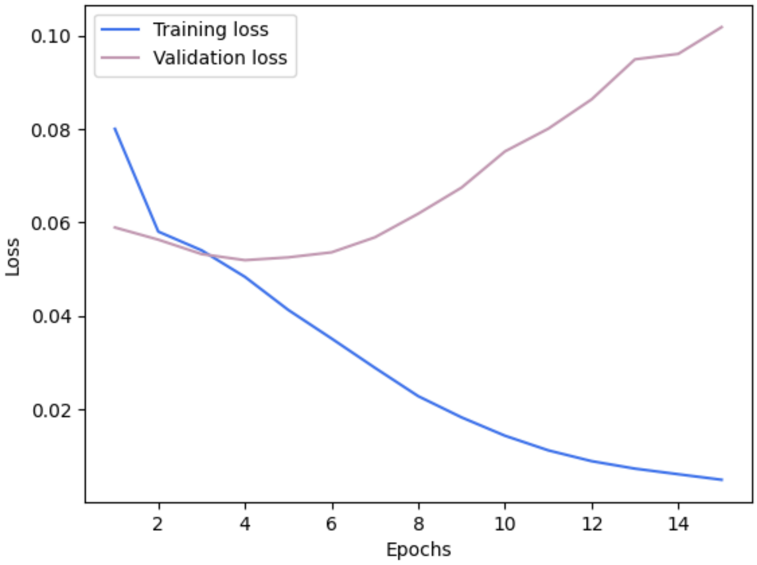


Figure B.1: *H6* model loss evolution, using FL.

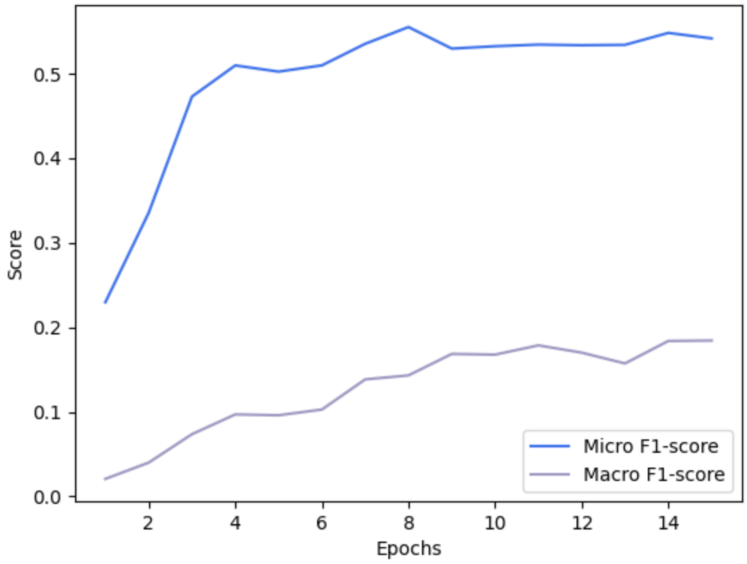


Figure B.2: *H6* model score evolution, using FL.

observe a clear disparity. The model using FL shows a substantially lower macro f1-score, both by the end of the 15 epochs as well as at the time of overfitting.

By plotting the evolution of the loss and score for the model using the BCE (figures B.3 and B.4 respectively) we observe a smoother curve, that slowly increases after 10 epochs, which makes it seem the closer to converge while still being far from it. Nonetheless, the performance in terms of macro f1-score is very poor, even after 15 epochs, when the model starts overfitting, which indicates the model cannot extract any underlying pattern for most of the classes apart from the most represented ones.

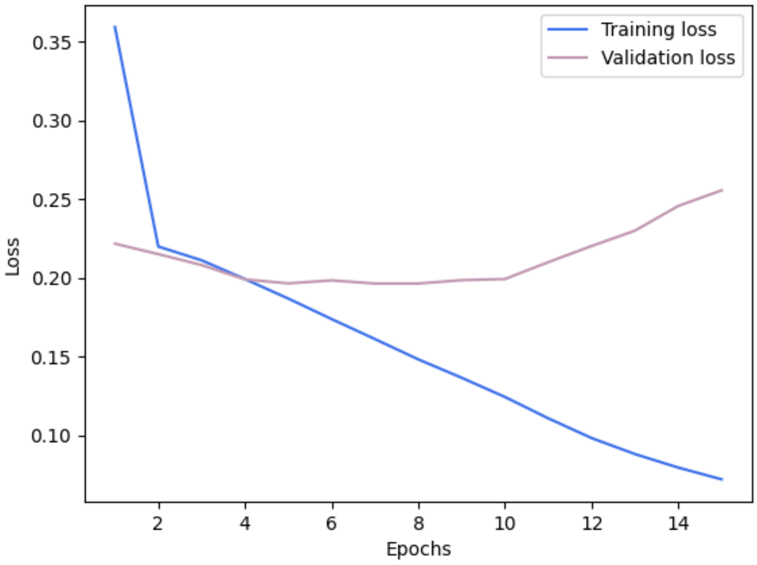


Figure B.3: *H6* model loss evolution, using BCE.

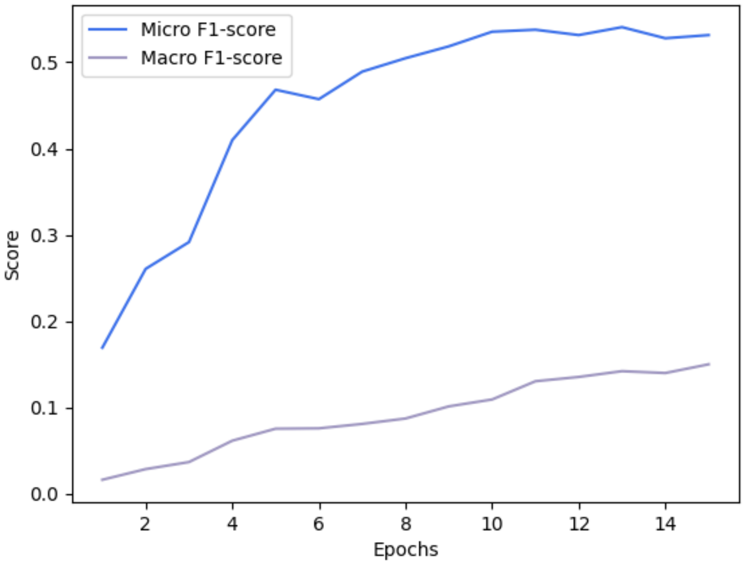


Figure B.4: *H6* model score evolution, using BCE.

Looking at the plotted graphs we conclude that there's no point in assuming ASL was a bad choice since it would still be preferable to train the model for only 5 epochs then to train the same model with FL and BCE with more epochs, as eventually both of them show signs of overfitting while having a worse performance.

## B.2 Using simple DistilBERT

One of the factors that can lead a model to over-fit is its complexity. Our model has some complexity because, on top of a pre-trained language model (DistilBERT) we add two linear layers, a dropout layer and a couple of convolutional layers and we train it, while also fine-tuning DistilBERT's pre-trained layers.

To understand the possible relationship between the extra complexity and the overfitting problem we trained a new model where we tried to remove said complexity. We were left with a simple DistilBERT model (with the two initial layers frozen) and a linear and a dropout layer.

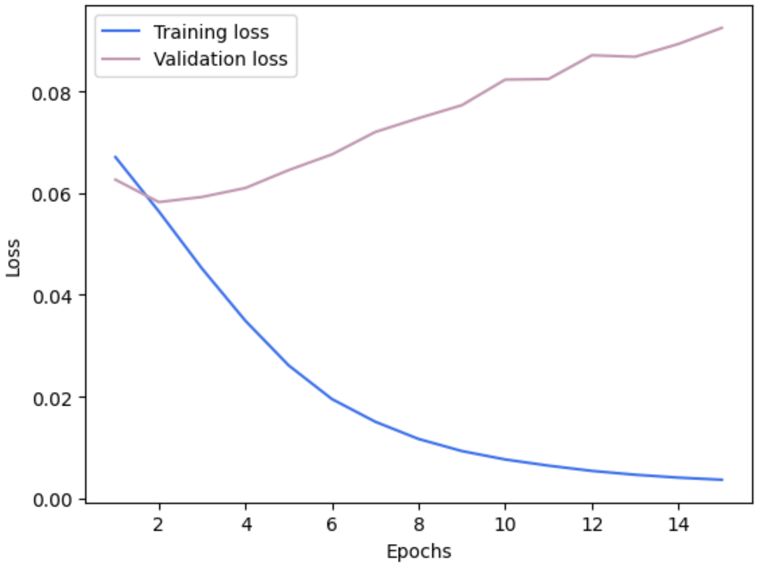


Figure B.5: Plain DistilBERT model loss evolution.

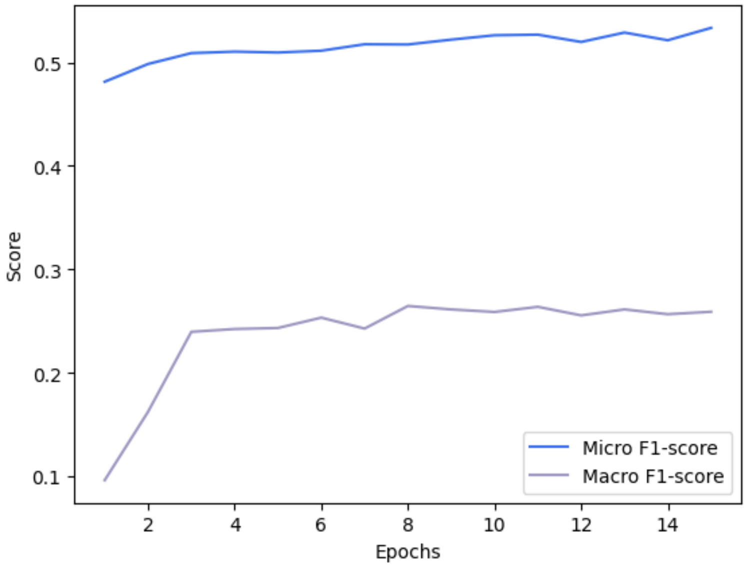


Figure B.6: Plain DistilBERT model score evolution.

The outcome of training a plain DistilBERT and plot its loss and score evolution is what we expected. The added complexity on model *H6* is not the reason for overfitting. In fact the signs of overfitting appear even earlier on this model which besides being simpler has a higher dropout rate (20%).

We went even further and made the model's training even simpler by freezing all DistilBERT's pre-trained layers except for the last one.

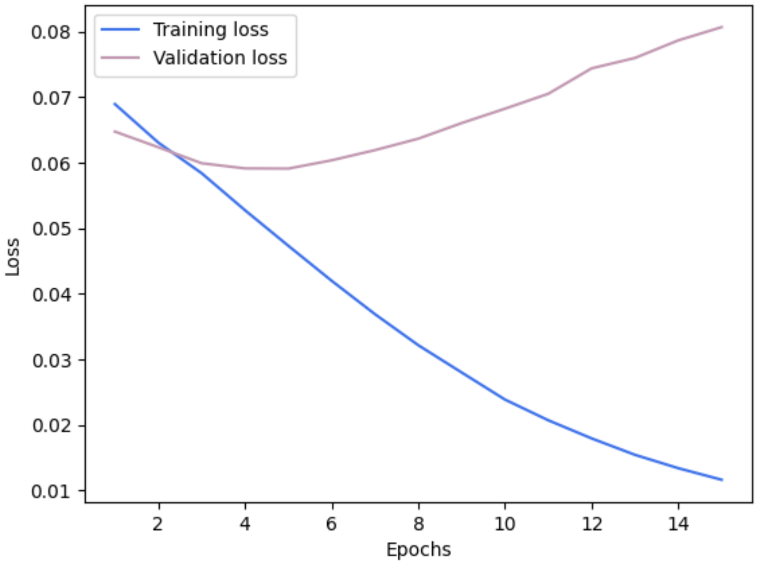


Figure B.7: Plain DistilBERT model, with 5 layers frozen, loss evolution.

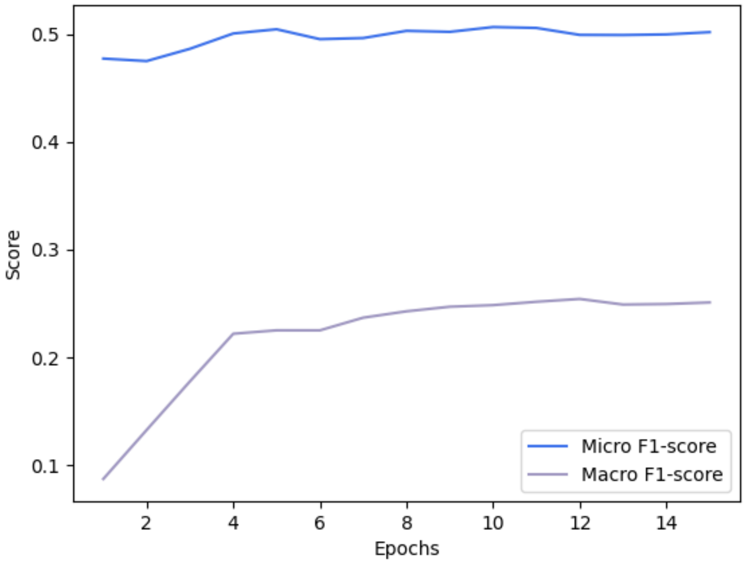


Figure B.8: Plain DistilBERT, with 5 layers frozen, model score evolution.

The results are very similar to the ones from the previous experience since after a couple of epochs the model starts overfitting. This leads us to believe that reducing the model's complexity or change its loss function is not the solution and that overfitting is probably caused due to the small number of training samples and its class distribution.

# Bibliography

[AG17] Neena Aloysius and M. Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592, 2017.

[AJO+18] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[ALA22] Stephen Akuma, Tyosar Lubem, and Isaac Terngu Adom. Comparing bag of words and tf-idf with different models for hate speech detection from live tweets. *International Journal of Information Technology*, 14(7):3629–3635, Dec 2022.

[Als21] Ibrahim Alshubaily. Textcnn with attention for text classification. *CoRR*, abs/2108.01921, 2021.

[ANK18] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, 1142(1):012012, nov 2018.

[AQA21] Dia Abujaber, Ahmed Qarqaz, and Malak A. Abdullah. LeCun at SemEval-2021 task 6: Detecting persuasion techniques in text using ensembled pretrained transformers and data augmentation. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1068–1074, Online, August 2021. Association for Computational Linguistics.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[BKD21] Ruurd Buijs, Thomas Koch, and Elenna Dugundji. Applying transfer learning and various ann architectures to predict transportation mode choice in amsterdam. *Procedia Computer Science*, 184:532–540, 2021. The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[Bod02] Mikael Boden. A guide to recurrent neural networks and backpropagation. *the Dallas project*, 2(2):1–10, 2002.

[BRZ+20] Emanuel Ben Baruch, Tal Ridnik, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. Asymmetric loss for multi-label classification. *CoRR*, abs/2009.14119, 2020.

[CDY+21] Yizhou Chen, Heng Dai, Xiao Yu, Wenhua Hu, Zhiwen Xie, and Cheng Tan. Improving ponzi scheme contract detection using multi-channel textcnn and transformer. *Sensors*, 21(19), 2021.

[CMM11] Everton Alvares Cherman, Maria Carolina Monard, and Jean Metz. Multi-label Problem Transformation Methods: a Case Study. *CLEI Electronic Journal*, 14:4 – 4, 04 2011.

[DAS+21] Dimitar Dimitrov, Bishr Bin Ali, Shaden Shaar, Firoj Alam, Fabrizio Silvestri, Hamed Firooz, Preslav Nakov, and Giovanni Da San Martino. Semeval-2021 task 6: Detection of persuasion techniques in texts and images, 2021.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[DCWZ16] Xuedan Du, Yinghao Cai, Shuo Wang, and Leijie Zhang. Overview of deep learning. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 159–164. IEEE, 2016.

[DGX18] Jiachen Du, Lin Gui, and Ruifeng Xu. *A Convolutional Attention Model for Text Classification*, pages 183–195. 01 2018.

[DSMYBC+19] Giovanni Da San Martino, Seunghak Yu, Alberto Barrón-Cedeño, Rostislav Petrov, and Preslav Nakov. Fine-grained analysis of propaganda in news article. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5636–5646, Hong Kong, China, November 2019. Association for Computational Linguistics.

[FOJ19] Angel Fiallos Ordoñez and Karina Jimenes. Using reddit data for multi-label text classification of twitter users interests. *2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG)*, 04 2019.

[GGM21] Kshitij Gupta, Devansh Gautam, and Radhika Mamidi. Volta at semeval-2021 task 6: Towards detecting persuasive texts and images using textual and multimodal ensemble. *CoRR*, abs/2106.00240, 2021.

[GS21] Vansh Gupta and Raksha Sharma. NLPIITR at SemEval-2021 task 6: RoBERTa model with data augmentation for persuasion techniques detection. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1061–1067, Online, August 2021. Association for Computational Linguistics.

[HGK+21] Yi Huang, Buse Giledereli, Abdullatif Köksal, Arzucan Özgür, and Elif Ozkirimli. Balancing methods for multi-label text classification with long-tailed class distribution. *CoRR*, abs/2109.04712, 2021.

[HR18] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[HXY15] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

[IB19] Muhammad Okky Ibrohim and Indra Budi. Multi-label hate speech and abusive language detection in Indonesian Twitter. In *Proceedings of the Third Workshop on Abusive Language Online*, pages 46–57, Florence, Italy, August 2019. Association for Computational Linguistics.

[ISB19] Muhammad Okky Ibrohim, Muhammad Akbar Setiadi, and Indra Budi. Identification of hate speech and abusive language on indonesian twitter using the word2vec, part of speech and emoji features. In *Proceedings of the International Conference on Advanced Information Science and System*, AISS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[JPY22] Praphula Kumar Jain, Rajendra Pamula, and Ephrem Admasu Yekun. A multi-label ensemble predicting model to service recommendation from social media contents. *The Journal of Supercomputing*, 78(4):5203–5220, Mar 2022.

[JT15] Kevin G. Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. *CoRR*, abs/1502.07943, 2015.

[KFM$^+$21] Douwe Kiela, Hamed Firooz, Aravind Mohan, Vedanuj Goswami, Amanpreet Singh, Pratik Ringshia, and Davide Testuggine. The hateful memes challenge: Detecting hate speech in multimodal memes, 2021.

[Kot13] Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.

[KSND21] Divya Khyani, BS Siddhartha, NM Niveditha, and BM Divya. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 2021.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LGG$^+$17] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[MBW$^+$20] Giovanni Da San Martino, Alberto Barrón-Cedeño, Henning Wachsmuth, Rostislav Petrov, and Preslav Nakov. Semeval-2020 task 11: Detection of propaganda techniques in news articles. *CoRR*, abs/2009.02696, 2020.

[MCCD13] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[Met] Meta. How facebook uses super-efficient ai models to detect hate speech.

[MFGA21] Nicola Messina, Fabrizio Falchi, Claudio Gennaro, and Giuseppe Amato. AIMH at SemEval-2021 task 6: Multimodal classification using an ensemble of transformer models. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1020–1026, Online, August 2021. Association for Computational Linguistics.

[Mit97] Tom Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[Moh18] Fahim Mohammad. Is preprocessing of text really worth your time for online comment classification? *CoRR*, abs/1806.02908, 2018.

[PK17] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. pages 189–204, 2017.

[Ras14] Sebastian Raschka. Naive bayes and text classification I - introduction and theory. *CoRR*, abs/1410.5329, 2014.

[Sch20] Katharine Schwab. We analyzed 1.8 million images on twitter to learn how russian trolls operate, Aug 2020.

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[Sem] SemEval. Semeval 2021 task 6 on "detection of persuasion techniques in texts and images".

[SJ19] Kamilya Smagulova and Alex Pappachen James. A survey on lstm memristive neural network architectures and applications. *The European Physical Journal Special Topics*, 228(10):2313–2324, 2019.

[Sor10] Mohammad S Sorower. A literature survey on algorithms for multi-label learning. 2010.

[STF21] Thorben Schomacker and Marina Tropmann-Frick. Language representation models: An overview. *Entropy*, 23(11), 2021.

[Suc21] Peter Suciu. Americans spent on average more than 1,300 hours on social media last year, Dec 2021.

[SYG$^+$20] Sheng Shen, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Rethinking batch normalization in transformers. *CoRR*, abs/2003.07845, 2020.

[TGL$^+$21] Junfeng Tian, Min Gui, Chenliang Li, Ming Yan, and Wenming Xiao. MinD at SemEval-2021 task 6: Propaganda detection using transfer learning and multimodal fusion. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1082–1087, Online, August 2021. Association for Computational Linguistics.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[WSC$^+$16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[WSM$^+$18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[YCD$^+$20] Lian Yu, Lihao Chen, Jingtao Dong, Mengyuan Li, Lijun Liu, Bai Zhao, and Chen Zhang. Detecting malicious web requests using an enhanced textcnn. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 768–777, 2020.

[ZQD$^+$19] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019.

[ZW17] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[ZWZ21] Xingyu Zhu, Jin Wang, and Xuejie Zhang. YNU-HPCC at SemEval-2021 task 6: Combining ALBERT and text-CNN for persuasion detection in texts and images. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1045–1050, Online, August 2021. Association for Computational Linguistics.

**Contactos:**
Universidade de Évora
**Escola de Ciêcias e Tecnologia**
7000 - 671 Évora | Portugal
Tel: (+351) 266 745 371
email: geral@ect.uevora.pt