Cross-lingual Toxicity Detection

Loïs Bilat

Master Thesis

École polytechnique fédérale de Lausanne
School of Computer and Communication Sciences

Supervisor at EPFL
Jean-Cédric Chappelier

Supervisors at ELCA Informatique SA
Peter Sperisen
Matthias Ramirez

August 2020

Abstract

With the increasing use of social media, there is a critical need for performant automatic moderation tools. In this thesis, we present advanced classifiers that can detect hateful and offensive content in short texts. We study various architectures based on transformer models such as BERT and evaluate multiple changes to those models that improve their performance. We then tackle cross-lingual classification and introduce new architectures that use joint-learning and data translation. Our models are able to outperform existing multilingual models on zero-shot and multilingual classification.

Acknowledgements

I would like to thank everyone without whom this project would not have been possible. To begin with, I want to thank Peter Sperisen and Matthias Ramirez, my supervisors at ELCA, for their continuous guidance and support during those 6 months, as well as the rest of the ELCA Data Science team for their warm welcome and the interesting conversations we had. I also wish to thank Jean-Cédric Chappelier for expressing interest in this project and accepting to supervise me, and for his valuable oversight of this thesis. Finally, I would like to thank my family and friends for all their unconditional support during these last five years.

Disclaimer

Because of the nature of this work, we need to work with offensive and hateful content directly, content that might be shocking and hurtful to some readers. Examples of such content are presented in the report for illustration purposes, and while the more offensive and obscene words have been partially censored, it is still quite easy to guess the original content. Most of the examples are taken directly from available datasets, but some were created specifically for this report. Those examples do not reflect the views of the authors.

Contents

1 | Introduction

1.1 | Motivation

With the rise in popularity of social media in the past decade, an increasing number of people are exposed to toxic and hateful comments on the Internet, a phenomenon often amplified by the anonymity provided by some platforms.[1] It is crucial for the moderators of these platforms to have an efficient way to filter and remove toxic content, for the well-being of the users, and the moderators themselves that are often left with psychological scars from the manual moderation of such content.[2]

In the past few years, we have seen increasing pressure on social media platforms to be better at removing hateful content, with some governments pushing for more regulations and accountability. A few countries have tried implementing laws to force social media platforms to report and remove hateful content or risk fines otherwise, but such initiatives are often very controversial and considered by their opponents as anti-constitutional and against freedom of speech.[3,4]

Fully manual moderation is not possible, for the psychological reasons cited earlier, but also due to the sheer amount of content posted every day. Automatic or semi-automatic moderation tools can be of huge help and make the Internet a safer place. The largest problem when developing such systems is to find enough data to train them. Unfortunately, there are not a lot of resources available, and the vast majority of obtainable data are in English, making this task even more difficult for other languages.

In this project, we will implement models that can automatically detect hateful and offensive content, and then try to transfer the knowledge acquired from one language to other languages with fewer or no data. We will study different categories of models, and starting from the best existing models we will work towards improving them and applying them successfully to a cross-lingual context.

1.2 | Problem Statement

We want to classify short messages into three categories: *hate speech*, *offensive content* and *none*. These messages can originate from various sources, such as social media, online chats or comments on websites. We are only interested in the content of the message, but not the metadata or other additional information.

[1] https://medium.com/@empathyprojectonline/whats-behind-our-bad-behavior-and-lack-of-empathy-online-8b63e46b76b8

[2] https://www.theguardian.com/technology/2019/sep/17/revealed-catastrophic-effects-working-facebook-moderator

[3] https://en.wikipedia.org/wiki/Netzwerkdurchsetzungsgesetz

[4] https://edition.cnn.com/2020/05/13/tech/french-hate-speech-social-media-law

We use the definitions from Charitidis et al. [1] for the three categories:

- `hate`: any message that fulfils the following two criteria: it should target a person or a group, and it must contain a hateful attack; a hateful attack can be violent speech, support for death, harm or disease, a statement of inferiority regarding a group they identify with or a call for segregation (e.g. racism, homophobia, Islamophobia, sexism);
- `offensive`: any message that contains a personal attack, harassment, or insults;
- `none`: any message that does not belong to one of the former two categories.

To illustrate these definitions, we show in table 1.1 a few examples of sentences taken from various hate speech and toxic English datasets.

`hate`	@user May he burn in hell eternally.
	@user Hang them and let god sort them out
	@user @user F*ck that n*gga
	@user Build the wall! Immigrants are bring in the disease
`offensive`	@user What a fat c*nt
	@user This low life will do anything to hang on to power!
	@user @user These people are idiots. Truly.
	ur trash!
`none`	@user @user @user @user I'm playing the lotto.
	@user @user pls cancel this appointment.
	RT @user: @user @user You win the Twitter award tonight!! IMHO.
	@user We're f*cking tired.

Table 1.1. Example of content from each class. Censorship was added for the report and is not present in the data. Mentions were simplified for clarity.

In other works on the same subject, similar definitions are often used, although some variations are not uncommon. The datasets selected for this project (presented in section 3.1) all use equivalent definitions.

1.3 | Objectives

There are two main objectives to this thesis:

Monolingual Toxicity Detection: The first objective is to implement performant models that can classify messages from a given language into one of the three classes presented in section 1.2. We will develop classifiers for English, French and German data using advanced Neural Network based techniques, and try to improve existing methods and architectures. Each model should perform well on all three classes. We present this work in chapter 4.

Cross-lingual Toxicity Detection: The second objective is to design multilingual models that can classify messages in multiple languages. They should perform similarly to monolingual models when evaluated on languages with training data available, but they should also work on languages without training data. This aspect is developed in chapter 5.

2 | Related work

There were three main steps in the development of toxic content classification; it started with simple models that were trained on individual datasets for the specific toxicity detection task, then transfer learning methods were introduced to leverage information from models already trained on more general data, and finally, multiple studies experimented with multilingual classification to transfer knowledge to languages with no training data.

2.1 | Toxicity Detection

Traditional toxic content detection methods are built on algorithms such as Logistic Regression and Support Vector Machines [2, 3, 4, 5]. A feature vector is created from the text, usually by simply considering the textual content, and encoding it into a vector (bag of words, tf-idf) using word n-grams as well as character n-grams [2, 3, 4, 5]. When available, external features can also be added (for instance with tweets, the number of mentions or retweets can be used [2]) and are usually quite useful, but too reliant on the type of data used and not applicable to text coming from more diverse sources. More advanced features have also been used [2], such as part-of-speech tags, Flesch-Kincaid Grade Level, Flesch Reading ease scores[1] and sentiment score. Unfortunately, such methods are very sensitive to the presence of some keywords and are not able to see long-term dependencies in the text, resulting in overall mixed results.

Neural Network approaches such as convolutional neural networks (CNN) and long short-term memory (LSTM) networks were also applied to the toxic content detection task [6], as these architectures are able to act as n-gram feature extractors. Mixes of convolutional and recurrent models were also tried, using a model were the output of a CNN was fed into a gated recurrent unit (GRU) layer [7]. Aken et al. [4] added attention layers to LSTM and GRU networks and experimented with ensemble learning. They also perform an in-depth error analysis to understand the primary causes of misclassification. Most errors come from ambiguous sentences (for instance with irony or sarcasm, metaphors, rhetorical questions or rare words), but a lot of errors also come from inconsistencies in the annotations, and the presence of some specific keywords that drastically influence the perceived offensiveness of the sentence. Charitidis et al. [1] compared multiple models on a newly created dataset, and Gambäck and Sikdar [8] experimented with different word embeddings for a CNN network. Using these types of models was for a long time the best way to perform toxic content detection, but newer methods based on transfer learning and more complex architectures are now consistently better. We will still take time to evaluate some of these models and use ideas from those papers to improve the newer architectures.

[1] https://en.wikipedia.org/wiki/Flesch-Kincaid_readability_tests

2.2 | Transfer Learning

Over time, the amount of available data for various natural language processing tasks rapidly increased, and the models utilising those data were getting more complex. To spare some resources, it is possible to reuse already trained components in the target model instead of training them completely from scratch. This is convenient to reduce the training time, but also to improve the quality of the model that will have a better language model representation.

Widespread use of pretrained components in natural language processing started with the introduction of pretrained word embeddings such as word2vec [9], GloVe [10] and FastText [11]. Those were trained on large amounts of data and could be used in a multitude of tasks, saving computation and time resources that would have been required to create completely new word embeddings. With those embeddings it became easier and quicker to create complex language models, as those models were able to detect similarities between words, unlike previous bag-of-word based methods (words with a similar meaning or use are often close to each other in the embedding space). However, they lack context awareness which causes some issues with homonyms that can have different meanings depending on their context.

With ELMo [12], a bidirectional LSTM network was used to create contextual word embeddings that could then be used for any language processing task. BERT (Devlin et al. [13]) followed with the same idea but used stacked Transformer encoders (Vaswani et al. [14]), which became state-of-the-art on eleven different language processing tasks. Many variants followed, such as RoBERTa [15], an optimised version of BERT trained with more data, or XLNet [16] that improves BERT by using permutations instead of masked tokens.

Iterating on models similar to BERT, various classifiers were added to extract important features [17]. This process is called fine-tuning, where during the training of the model the added classifier and the BERT model are updated simultaneously to better fit the data. Diverse pretraining methods were used to improve the relevance of the BERT model before using it for fine-tuning [18]; it has been shown that continuing the original training on data relevant to the target task can be useful in some cases.

Since the introduction of BERT, lots of researchers have focused on improving the original version of BERT, either by developing new models based thereon or by finding better ways to use the embeddings created by it. Large research teams and companies regularly come up with new models based on BERT, with more and more parameters, training data, and optimisations.[2] In this project we will study the second approach, and using some ideas from previous research will try to discover better ways to utilise pre-trained models.

2.3 | Multilingual toxicity detection

Most research in natural language processing is carried out on English data, and it is hard to find resources for other languages. Multilingual text classification was developed to allow for the creation of models in languages with few or no training data without the need for manual data annotation, a costly process. There are two main approaches to this problem:

Translation-based methods: The first approach is to translate all the data into a single language with many resources, and then use a traditional monolingual model [20]. Some more advanced models that use two-way translations to train multiple models in parallel have also been successfully used [21, 22].

[2] Recently, the OpenAI team presented GPT-3, a language model with 175 billion parameters, compared to BERT's 110 million. It is to our knowledge the largest language model to this date [19].

Multilingual methods: The other method is to use multilingual embeddings, so that we can train a model that can comprehend multiple languages simultaneously, such as LASER embeddings [23] that are able to create sentence embeddings for 93 languages, all residing in the same embedding space, and MUSE embeddings [24], aligned versions of the FastText word embeddings [11] in 30 languages. A multilingual version of BERT was also created,[3] and later other models with similar architectures like XLM [25] and XLM-RoBERTa [24].

The choice of the approach often depends on the language and the actual datasets used [20]. No method is consistently better than the others. Both approaches will be evaluated in this project in chapter 5.

[3] https://github.com/google-research/bert/blob/master/multilingual.md

3 | Background

3.1 | Data

Gathering data for toxic comment classification is a challenging task, as manual annotation is required and it is not trivial to gather toxic data in the first place. Hateful comments are quite sparse online, especially since some of them are fortunately rapidly removed, either by the author or a moderator, leading to a large class imbalance with naive data gathering. Still, there have been numerous attempts in the past few years to collect such data.[1] Most data originate from Twitter, and this is the type of content that we use in this project.

There are, however, some common problems with the available data. The majority of research on the subject has been done on the English language, and resources in most other languages are sparse or non-existent, complicating the creation of models. Besides, the annotation scheme often differs from one dataset to another; the number of classes can vary, sometimes a binary classification between toxic and non-toxic comments is performed, or sometimes hateful content is also further annotated into distinct categories. We conducted an exhaustive search of available datasets and present our selection in the following section. This selection is based on the size of the dataset (we excluded small datasets when we already had multiple larger datasets in one language), the annotations available (they had to be compatible with the `hate`, `offensive` and `none` classes of our problem, see section 1.2), and a clear documentation on the source and the acquisition methodology of the data.

3.1.1 | Available datasets

Here, we present the datasets that were chosen for this project. We have datasets in English, French and German that are used in both the first part (*Monolingual Toxicity Detection*, chapter 4) and the second part (*Cross-lingual Toxicity Detection*, chapter 5) to train some models. There are also additional datasets in other languages (Spanish, Italian, Turkish, Greek and Indonesian) that are only used in the second part to evaluate multilingual models, but not for training.

Dachs

The first dataset, published by Charitidis et al. [1] for DACHS (*A Data-driven Approach to Countering Hate Speech*), contains annotated tweets in five languages. We focus on the English, French and German data, but also use the Spanish and Greek data in the second part of the project. For each language two datasets are provided, one annotated for hate speech, the other annotated for personal attack, and for each dataset they provide tweet ids and a binary classification (*hate or not* for the first dataset, or *attack or not* for the second dataset). Each tweet was annotated by one person only, and the majority of tweets appear in both the *hate* and *attack*

[1] An extensive collection of such datasets can be found on http://hatespeechdata.com

datasets. We used the Twitter API[2] and the Twython library[3] to retrieve the text associated to these ids. However, some of those tweets have since been deleted (approximately 30%), we were therefore unable to retrieve all the data corresponding to the original datasets.

The datasets were then adapted to our problem statement to match the previously defined classes (`hate`, `offensive` and `none`). For each language, both datasets were joined and each tweet tagged as *hate speech* in the original dataset is tagged as `hate` in our dataset. Each tweet that is marked as *personal attack* but not *hate speech* in the original dataset is classified as `offensive` in our dataset. Finally, each tweet that is neither a personal attack nor a hate speech is classified as `none` in our dataset.

Founta et al.

The second dataset is provided by Founta et al. [26] and consists of English tweets, each annotated by five people, but only the tweet ids are available. Since it is already provided as a multi-class problem, no further transformation was required. In this dataset, the possible classes are *hate speech* (corresponding to our `hate` class), *abusive language* (`offensive`), *neutral* (`none`), and *spam*. We removed all spam messages (there are only a few of them) as we cannot assign them to any of the three classes we are interested in.

Davidson et al.

The third dataset, provided by Davidson et al. [2], is also a multi-class dataset in English with labels *hate*, *offensive* and *neither*, corresponding to our three classes. The tweets are annotated by at least three people each and are directly provided.

Jigsaw

The fourth dataset was published by Jigsaw and Google on Kaggle for a machine learning challenge.[4] This data, in English, comes from Wikipedia comments and contains among others the following annotations: *severe toxicity*, *identity attack*, *threat* and *insult*. Each of those is a number between 0 and 1, representing the average between different annotators that had to label 0 (no) or 1 (yes) for each category. Messages with value *identity attack* larger than 0.5 or value *threat* larger than 0.5 are labelled as `hate`, messages with value *insult* larger than 0.5 or value *severe toxicity* larger than 0.5 are labelled as `offensive`, if they are not already in the `hate` class, and the rest goes into the `none` class.

Moreover, because of the large quantity of data in the `none` class compared to the other two classes, it was decided to under-sample data from this class. We arbitrarily chose to keep 100'000 messages from the `none` class, which was then slightly reduced when removing some duplicate messages, giving us a proportion of `hate` messages similar to the other datasets (around 5%), while having approximately 50% of toxic messages.

In a more recent challenge proposed on Kaggle by the Jigsaw team,[5] an evaluation set with data in Spanish, Italian and Turkish is provided. The annotation only consists of a binary label indicating the presence or not of toxicity, but we will still evaluate our models from the second part of this project on this data.

[2] https://developer.twitter.com/en/docs
[3] https://twython.readthedocs.io/en/latest/
[4] https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification
[5] https://www.kaggle.com/c/jigsaw-multilingual-toxic-comment-classification

MLMA

The fifth dataset, provided by Ousidhoum et al. [3], contains data in English, French and Arabic. It consists of tweets that are annotated as *offensive*, *disrespectful*, *hateful*, *fearful*, *abusive* or *normal* (6 classes), with three annotators for each message. English and Arabic data were ignored, as the English dataset was extremely small and obtained very bad results compared to the other English datasets, and we are not interested in Arabic data. Any tweet that was labelled as *hateful* or *fearful* was considered as `hate`, any tweet labelled as *abusive* or *offensive* was considered as `offensive`, and the remaining tweets as `none`.

Germeval

For the sixth dataset, we regrouped data from the GermEval2018[6] and GermEval2019[7] challenges. For both challenges, we mapped the *abuse*, *insult* and *other* labels to `hate`, `offensive` and `none`, respectively, and we concatenated both datasets.

Indonesian Hate Speech

The final dataset, provided by Ibrohim and Budi [27], consists of Indonesian tweets with labels for *hate speech* and *abusive language*, that we map to `hate` and `offensive`. This dataset is only used in the second part of this project to evaluate multilingual models.

Other datasets

Other datasets were also available but were excluded for multiple reasons, often due to an incompatibly with our annotation scheme, for instance with only a binary classification scheme or missing classes [5, 28, 29]. Additional datasets can be found on *hatespeechdata*.[8]

Summary

We summarise all available training data in figure 3.1 and table 3.1.

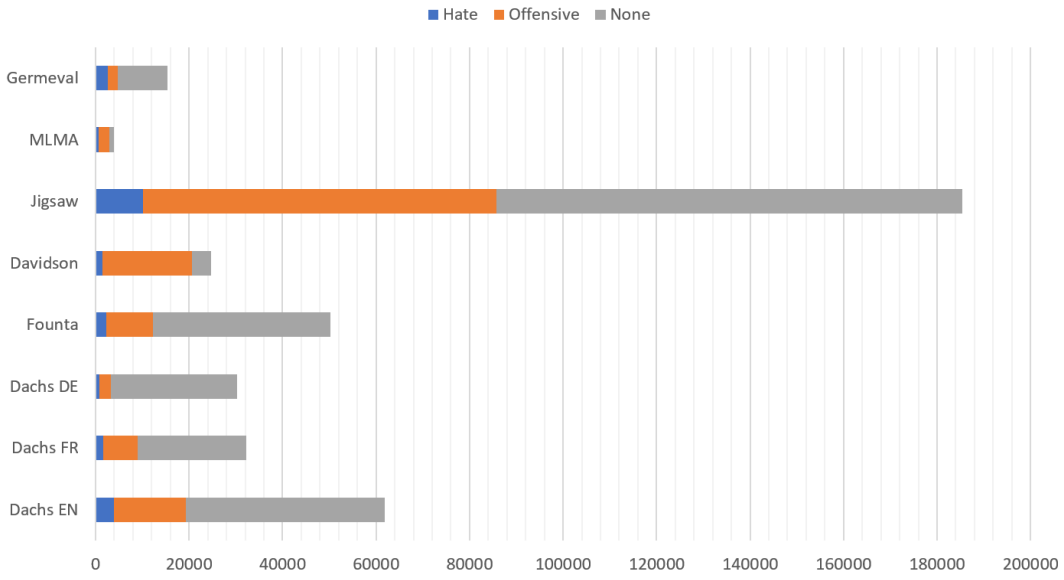

Figure 3.1. Summary of all the available data in English, French and German.

[6] https://github.com/uds-lsv/GermEval-2018-Data
[7] https://projects.fzai.h-da.de/iggsa/projekt/
[8] http://hatespeechdata.com/

		Count				Ratios		
Dataset	lang	Hate	Offensive	None	All	Hate	Offensive	None
	en	3'906	15'450	42'471	61'827	6.3%	25.0%	68.7%
Dachs	fr	1'716	7'337	23'218	32'271	5.3%	22.7%	71.9%
	de	779	2'524	27'014	30'317	2.6%	8.3%	89.1%
Founta	en	2'319	9'987	37'897	50'203	4.6%	19.9%	75.5%
Davidson	en	1'430	19'190	4'163	24'783	5.8%	77.4%	16.8%
Jigsaw	en	10'201	75'467	99'659	185'327	5.6%	40.7%	53.7%
MLMA	fr	733	2'281	993	4'007	18.3%	56.9%	24.8%
Germeval	de	2'665	2'046	10'707	15'418	17.3%	13.3%	69.4%
	en	17'856	120'094	179'340	317'290	5.6%	37.8%	56.5%
Total	fr	2'449	9'618	24'211	36'278	6.8%	26.5%	66.7%
	de	3'444	4'570	37'721	45'735	7.5%	10.0%	82.5%
All		25'213	137'668	246'862	409'743	6.2%	33.6%	60.2%

Table 3.1. Summary of all the available data in English, French and German, with the class distributions.

English is the language with the most resources as expected, and the toxic classes are in minority. While this represents the inherent distribution of this type of data, it also means it will be harder to learn a good classification for those lower resource classes, and the metric will need to be selected accordingly.

In addition to the English, French and German data that will be used to train our models, we have data that will exclusively be used in the second part of this project to evaluate the performance of our multilingual models on unseen languages. We show a summary of available datasets for these languages in table 3.2.

	Count				Ratios		
Dataset	Hate	Offensive	None	All	Hate	Offensive	None
Dachs GR	879	10514	41764	53157	1.6%	19.8%	78.6%
Dachs ES	676	1285	28114	30075	2.2%	4.3%	94.5%
Indonesian	5505	1743	5783	13031	42.2%	14.4%	44.4%
		Toxic	None	All		Toxic	None
Jigsaw Spanish		422	2078	2500		16.8%	83.2%
Jigsaw Italian		488	2012	2500		19.5%	80.5%
Jigsaw Turkish		320	2680	3000		10.7%	89.3%

Table 3.2. Summary of all data used only for evaluation of the multilingual models.

We observe a similar class imbalance as with the training datasets, but it is not a problem since we will not train any models on these datasets, as long as the metrics properly report the performance of the model on all classes.

3.1.2 | Data analysis

Before applying any model to the data, it is important to analyse more attentively the content of each dataset. To be specific, we are looking for recurrent keywords and themes in each class, which could let us know if there are considerable differences between the datasets. This analysis is performed on the English, French and German datasets.

For each dataset, a feature vector is generated for each sentence using a simple bag of unigrams approach. A Logistic Regression Classifier from the scikit-learn[9] library [30] is used to train a classification model, and then we extract the features with the highest coefficients. We remove stopwords and only keep one member of each word family (words with the same lemma or alternative spellings). We show in table 3.3 for each dataset some of the most important words for the `hate` and `offensive` classes.

Dataset	Class	Dataset
Dachs EN	Offensive	cretins, bullsh*t, hypocrite, rag, moron
	Hate	c*on, castrate, hang, punch, rot
Founta	Offensive	f*cked, sh*t, bullsh*t, p*nis, p*ssy
	Hate	n*gga, terrorists, hate, females, nerds
Davidson	Offensive	b*tch, p*ssy, hoe, sh*t, c*nt
	Hate	f*ggot, n*gger, wetback, spic, dyke
Jigsaw	Offensive	idiot, stupid, hypocrite, moron, dumb
	Hate	n*gger, negro, black, f*ggot, gay
MLMA-FR	Offensive	mongol, attardé, origine, peur, jeunes
	Hate	expulser, rigole, noirs, chien, islamo
Dachs FR	Offensive	micron, duc*n, blaireau, torchon, c*nnasse
	Hate	pendre, creve, n*gros, b**gnoule, noyer
Dachs DE	Offensive	lugenpresse, *rschloch, ferkel, goldstucke, verrater
	Hate	verrotten, hinrichten, verrecken, invasoren, abfaulen
Germeval	Offensive	murksel, *rschloch, heuchler, dumm, pfeife
	Hate	abschaum, invasorien, asylanten, lugenpresse, moslems

Table 3.3. Most contributing words in each dataset for the `hate` and `offensive` classes, according to the logistic regression coefficients.

We see that the themes in each dataset are quite different. Hateful content in the Dachs EN datasets is about ways to hurt and kill other people, while offensive content mostly seems to be about "fakenews" and politics. For the Davidson and Founta datasets, `hate` mostly consists of comments, whereas `offensive` is mostly offensive language and insults. With the Jigsaw dataset, we have a mix of the previous distributions, mostly racist comments in `hate` and some lighter insults in `offensive`. In French, MLMA-FR tends towards racism while Dachs FR is more about ways to hurt others. The two German datasets are globally similar. To illustrate this difference in content, we show in figure 3.2 wordclouds consisting of the most important words for the `hate` and `offensive` classes, this time with all data from each language merged. We can see that with the datasets merged, the most important words are quite different. In all three languages, most of the important words for `offensive` were not present when considering the datasets separately, indicating that the model had to find more general words that were present in all datasets.

[9] https://scikit-learn.org

Due to this disparity in the type of content covered by each dataset, we will first study them separately, as there might be a significant domain gap that might cause problems when combining them.

Figure 3.2. Wordclouds of important words in the English, French and German datasets, according to the logistic regression coefficients.

3.2 | Evaluation Metrics

An important consideration is the choice of the evaluation metric for the considered task. One major problem with the data available, both in the datasets we use in this project and real-world applications, is the large class imbalance. Hateful content is underrepresented, but it still needs to be correctly accounted for in the metric. We preferred a known metric, to allow better reproducibility and comparison with previous and future work. For these reasons, we chose the macro-F1 score, the unweighted average of the F1-score of each class, where the F1-score for a class is the harmonic mean of the precision and recall on that class.

This metric was chosen for its property of giving as much importance to the minority classes as to the majority classes. Other metrics such as accuracy, weighted-average F1-score or micro-F1 accord more importance to the majority classes, and any sudden drop of performance on a minority class would have almost no influence on the final score. With the macro-F1, a performance drop on the `hate` or `offensive` class will have as much impact as a drop of performance on the `none` class. For some models, we will also report the recall and precision on all classes, as it gives more insight into the strengths and weaknesses of each model.

Using simpler metrics like the recall on the toxic classes was also considered, but those often had the problem of ignoring one aspect of the problem. Additionally, some extreme cases with very low precision could happen, and we were unable to automatically detect them with these simpler metrics. Experiments were carried out with a custom metric tailored for this problem, its usefulness is discussed in section 6.3.

Finally, before using any metric, it is important to identify some of its limits. It is especially interesting to know about results in extreme cases to define clear baselines. The Dachs EN dataset is used here to illustrate this aspect, but the conclusions are similar with the other datasets. We consider some naive models: three of them always predict the same class, `none`, `offensive` or `hate`, one makes a uniformly random classification, and the last one generates random predictions, but with probabilities proportional to the prior distribution of the classes. We report the precision and recall for all classes, with the macro-F1 score, in table 3.4. Every metric reported here and in this report is in percents.

		None		Offensive		Hate	
Model	m-F1	P	R	P	R	P	R
Always `none`	27	**68**	**100**	0	0	0	0
Always `offensive`	13	0	0	25	**100**	0	0
Always `hate`	30	0	0	0	0	**7**	**100**
Uniform random	28	**68**	33	**26**	35	6	42
Weighted random	**33**	**68**	68	**26**	26	**7**	7

Table 3.4. Metrics for naive classifiers on the Dachs EN dataset. In bold, the maximal value obtainable for each metric using one of the naive classifiers.

Two main conclusions can be drawn from those results. First, it is possible to achieve a macro-F1 score of 33% with a naive model that does not actually do anything useful. It is important to keep this in mind when experimenting with various models, as a score around this value that might initially indicate our model is learning something relevant actually does not mean anything. Secondly, we can see here why a score based uniquely on one metric (for instance the recall on `hate`) can be problematic, as it can reach maximal values with dummy models. The macro-F1 score does not suffer from this problem.

3.3 | Tokenisers

3.3.1 | Byte-pair encoding

Byte-pair encoding [31] is a subword tokenisation scheme that tries to optimise the amount of data required to represent text. It works as follows:

1. prepare a large corpus, and define a desired subword vocabulary size;
2. get the word count frequency and the initial token count (i.e. the number of occurrences for each character);
3. merge the two most common tokens, add the results to the list of tokens, remove the used tokens, and recalculate the frequency count for all tokens;
4. repeat until the vocabulary size is reached.

3.3.2 | WordPiece tokeniser

WordPiece tokenisation [32] is very similar to Byte-pair tokenisation, in the sense that it uses the same algorithm, except that instead of merging tokens based on their frequency, we merge them based on the increase in likelihood if merged. This difference is the probability of the new merged pair occurring, minus the probability of both tokens occurring individually.

4 | Monolingual Toxicity Detection

In this chapter we present our work on monolingual toxicity detection. We use the datasets individually and create some classifiers for each of them separately. Multiple architectures are presented, and potential improvements are tested. The combination of multiple datasets from the *same* language is also briefly studied.

4.1 | Models

There exist different types of classifiers, with various levels of complexity. For this problem, we experimented with three categories of models. We first have a simple statistical model, here a Logistic Regression classifier, easy to use and with a fast training time. Then there are classical neural networks, still with relatively simple architectures, but with a more consequent training time. Finally, there are so-called *transformer* models, large pretrained language models that are currently the state-of-the-art in most natural language processing tasks[1] but require significant training time. We primarily focus on this type of model in this project.

4.1.1 | Logistic Regression

As a first baseline, a Logistic Regression classifier from the scikit-learn[2] library [30] was chosen. For the features, we used count vectorisation on unigrams and bigrams of words, only keeping the 20'000 most frequent features.

The Logistic Regression classifier algorithm used here uses the "one versus rest" approach, where each class is treated as a separate binary classification problem. This approach was chosen as it is the default one in the scikit-learn library.

4.1.2 | Convolutional Neural Network

We implemented a simple convolutional neural network (CNN) using the PyTorch[3] library [33]. Each word in the input is first mapped to a vocabulary id. Using the Gensim[4] library [34], we load pretrained GloVe [10] or FastText [11] word embeddings, which are then loaded into the embedding layer provided by PyTorch. This embedding layer will map each vocabulary id to an embedding vector and can tune those embeddings during training. This can be useful since the type of data used to create the embeddings might be quite different from the data used for this project, both semantically and syntactically, hence the existing embeddings might need to be adjusted for optimal performance.

[1] State-of-the-art models with scores and associated papers are referenced on https://paperswithcode.com/area/natural-language-processing. In many natural language processing tasks, the best model is usually a transformer model

[2] https://scikit-learn.org

[3] https://pytorch.org/

[4] https://radimrehurek.com/gensim/

The embeddings outputted by the embedding layer are then sent into three 2D convolutional layers, each with one hundred filters of size 3,4 or 5 in the sentence length dimension and of embedding size in the other dimension. A rectified linear unit (ReLU) is applied, followed by a max-pool operation along the sentence dimension, and the resulting outputs from all three sizes are concatenated. We end with a dropout layer with probability 0.1, and a final linear layer that outputs three values, one for each class. The class with the highest value in the output is the predicted class. A softmax layer is applied by the implementation of the loss function to generate a probability for each class, therefore we do not include it ourselves. This architecture is illustrated in figure 4.1.

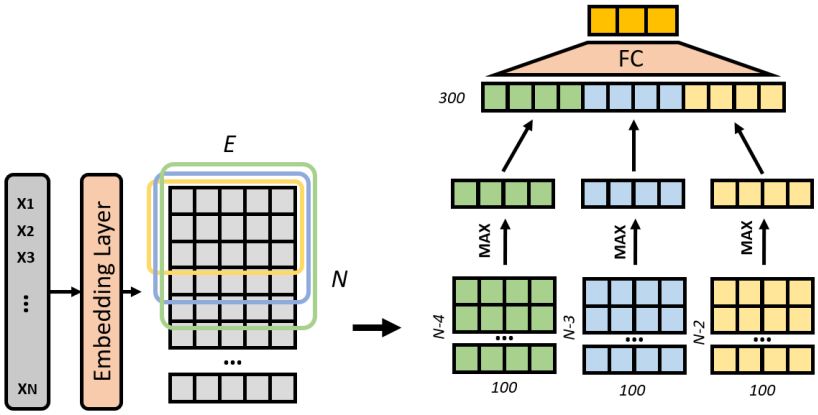

Figure 4.1. CNN architecture, for a sentence of length N, a word embedding of size E and 100 channels for each filter. The yellow, blue and green arrays correspond to the outputs of the three convolutions with kernel size 3,4 and 5 respectively.

4.1.3 | Bidirectional Long short-term memory network

We also implemented a model that uses two stacked bidirectional long short-term memory (biLSTM) layers. In a biLSTM, we have two classical LSTM layers, one going from the start of the sentence to the end, and the second going from the end to the start. The input is given to both networks at the same time, and then the hidden outputs of the two layers are concatenated. We subsequently reduce those to one vector by computing the mean of all the outputs, and to another vector by taking the maximum value for each dimension. We concatenate these two vectors, apply some dropout with probability 0.1, send this into a linear layer with 64 outputs, a ReLU layer and finally a linear layer with three outputs.

The embeddings are created by loading GloVe [10] or FastText [11] word embeddings with Gensim [34], and then using the PyTorch [33] embedding layer. The softmax layer is excluded here as it is included in the implementation of the loss function. This architecture is illustrated in figure 4.2.

4.1.4 | Transformer Models

Regarding *transformer models*, we use huggingsface's transformers[5] library [35], that let us use pretrained transformer models such as BERT [13], RoBERTa [15] and XLNet [16]. It provides a common interface to easily use and modify all of these models.

[5] https://huggingface.co/transformers

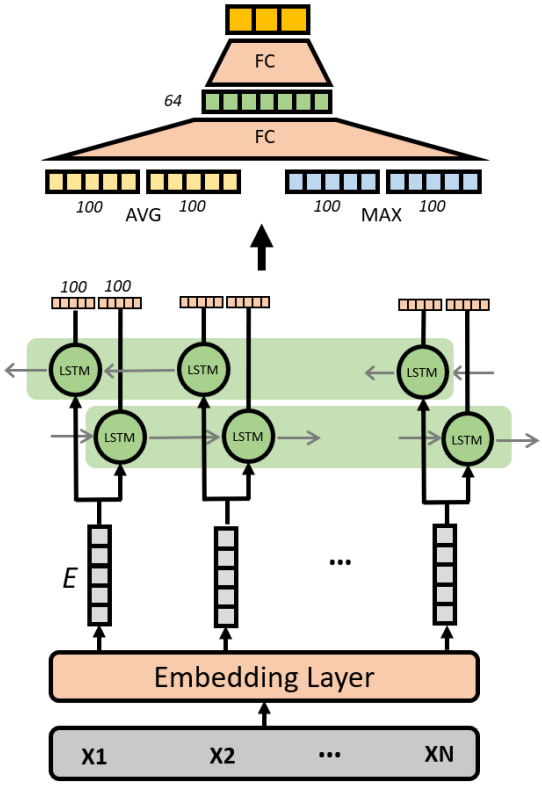

Figure 4.2. biLSTM architecture, for a sentence of length N, a word embedding size of E and a hidden size for the LSTM units of 100.

BERT

BERT (*Bidirectional Encoder Representations from Transformers*) was introduced by Devlin et al. [13] in 2018. It is able to create deep bidirectional contextual embeddings using raw unsupervised text data for training, that can then be used for numerous language processing and understanding tasks. There are two main phases when using a BERT model, or other similar transformer models: *pretraining* on general data and *fine-tuning* for a downstream task.

Pretraining: The pretraining step is typically performed by the authors of the model, as it requires a huge amount of computation power and a lot of data. BERT was trained on two unsupervised tasks, Masked Language Model (MLM) and Next Sentence Prediction (NSP). For the MLM task, a percentage of the input words are replaced by a `[MASK]` token, and the model is trained to predict the missing words. More precisely, 15% of the input tokens are selected, and among them 80% are replaced by the `[MASK]` token, 10% with a random token and the remaining 10% use the original word. For the NSP task, BERT will learn to predict the probability of two sentences being consecutive. During training, it will receive in 50% of cases two consecutive sentences from the training data and in the other 50% two nonconsecutive sentences at random, and it will learn to detect consecutive sentences.

Fine-tuning: In the second part, fine-tuning on the downstream task, the pretrained model is loaded and slightly adapted, usually by adding one or more new layers(s) extracting information from the transformer, constructing a network that will then be trained on the target task and data. During training, the weights of the added layer(s) as well as the weights from the original BERT network will be updated, learning a classifier that utilises those contextual embeddings, while also learning a better representation of the sentence by updating the inherent language model to be more adapted to the target domain.

BERT architecture

The input sentences are first converted to a specific input format. With BERT, the WordPiece tokeniser [32] is used to create the input tokens (see section 3.3.2), then a `[CLS]` token is added at the start of the sentence, and a `[SEP]` token is inserted between the first and the second sentence. For our problem, since we do not work on a task requiring a distinction between two sentences (this distinction is useful with question answering for instance), all sentences in the input are concatenated together, and the `[SEP]` token is added at the very end of the message. We then pad the input with `[PAD]` tokens, up to a length of 50 tokens. These tokens are then mapped to embeddings using an existing vocabulary-embedding mapping, creating *token embeddings*. We had to limit the maximum sentence size due to memory constraints, 50 was chosen as a good trade-off on our machine,[6] where we are able to run the desired models and the majority of sentences do not need to be shortened. It also was the value chosen by Charitidis et al. [1] as their maximum sentence size, and they noticed that larger sentences did not really improve performance. Assuming a maximum length of 25 tokens for the example, and using simple whitespace and punctuation-based tokenisation for convenience, a typical input using BERT would in our case look likes this:

```
[CLS] this is a very [MASK] car that you have .  I [MASK] that mine was
as nice .  [SEP] [PAD] [PAD] [PAD] [PAD] [PAD]
```

Additionally, BERT automatically creates *segment embeddings* and *position embeddings*. Segment embeddings are useful to differentiate two sentences in the case where we decide to separate them with a `[SEP]` token. A segment embedding is used for all tokens in the first sentence, and a different segment embedding is used for all tokens in the second sentence. Position embeddings are used to differentiate two instances of the same word but at different locations in the text. Two words, in two different instances, that are at the same position will have the same position embedding. These are required as the BERT architecture cannot understand positions by itself. An element-wise addition is then performed between the token embeddings, the segment embeddings and the position embeddings, giving us the *input embeddings*, as shown in figure 4.3.

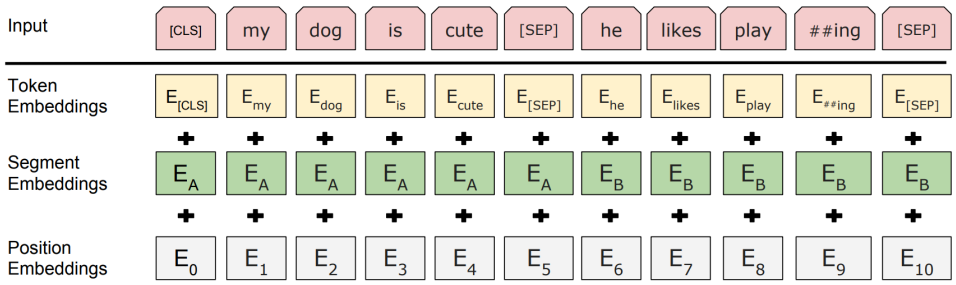

Figure 4.3. Bert input embeddings, illustration from Devlin et al. [13]

The rest of the architecture is a series of Transformer encoder blocks [14], illustrated in figure 4.4. The *input embeddings* are sent into the first Transformer encoder block, whose outputs are then forwarded to the next block. BERT uses 12 or 24 of these, depending on the exact model (BERT-base or BERT-large).

The first part of this encoder block is the *attention layer* that gives more importance to some parts of the input. This concept can be formulated using *queries*, *keys* and *values*. For some query, we want to know which keys are relevant, so that we can mask some irrelevant values

[6] NVIDIA GeForce GTX 1080Ti with 12GB of VRAM, 32GB of RAM

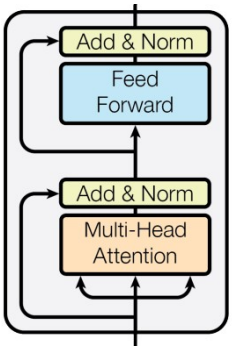

Figure 4.4. A Transformer encoder block, illustration from Vaswani et al. [14]

for each query. In our case, each word is a query, and we want to know how relevant the other words (keys) are to our main word/query. For this, we use weights on the values of words to give them some importance. An example of this concept is shown in figure 4.5.

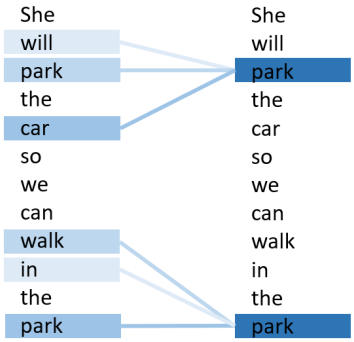

Figure 4.5. An illustration of the attention mechanism. On the left the different keys, on the right the different queries, and in blue the weights on the values. The attention of the first instance of the word *"park"* is focused on the word *"park"*, but also on *"car"*, giving some information about the meaning of the word *"park"*, and "*will*", giving the information that it is a verb. In the second instance of "*park*", with a different meaning, the attention is focused on other terms: "*walk*", "*in*" and "*park*". With an attention mechanism like this, the embeddings of the two instances of "*park*" should be significantly different, therefore creating *contextual* word embeddings.

All of the attentions can be computed in parallel if we represent the input as a matrix of shape (embedding size, sentence length). This matrix is used for the queries, the keys and the values (Q, K and V). We can then compute the attentions using the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^\intercal}{\sqrt{d_k}})V$$

$QK^\intercal$ computes the similarity between the queries and the keys, which is scaled by the root of the embedding size to avoid problems with a large embedding size. This similarity is then applied to the values to give some of them more or less importance (see figure 4.6, illustration on the left).

In the original Transformer encoder block, eight attentions are used for each query instead of only one, allowing the model to concurrently explot information from different representation spaces. A multi-head attention is used, where Q, K and V are first transformed into eight

different matrices each, using linear transformations, before going through the scaled dot-product attention. The resulting outputs are concatenated and sent to another linear transformation. Formally,

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_8)W^O$$
$$\text{with head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

The projections W_i^Q, W_i^K, W_i^V and W^O are parameter matrices, and they project the Q, K and V matrices into an eight times lower dimension space so that after concatenation the output has the same size as the input (see figure 4.6, illustration on the right).

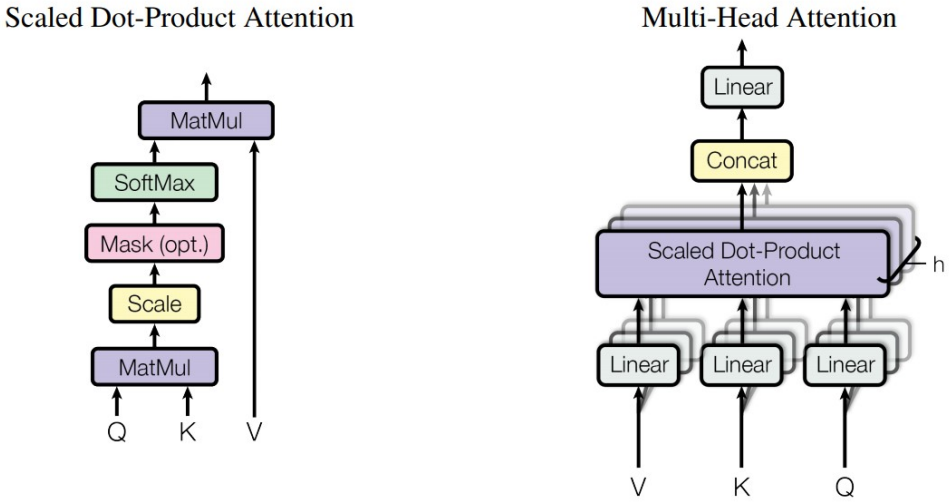

Figure 4.6. The attention module, image from Vaswani et al. [14]

The second module of this transformer block is the feed-forward layer, where two fully connected layers are applied to each position separately and identically, with a ReLU in between. The size in the middle of the two layers is four times the size of the input and output. Those can be seen as convolutions with a kernel size of one.

For each module (attention and feed-forward), a normalisation layer is added afterwards, taking the output from the module and a skip connection coming from before the module, concatenating and normalising them.

BERT [13] was trained on the BooksCorpus [36] and English Wikipedia,[7] for a total raw text size of approximately 16GB. Two main versions of BERT are available:

- BERT-base: 12 layers, hidden size of 768, 12 attention heads, 110M parameters
- BERT-large: 24 layers, hidden size of 1024, 16 attention heads, 340M parameters

The large version performs better than the base version, but we use the base version due to memory constraints.

[7] https://dumps.wikimedia.org/

BERT Variants

Since the introduction of BERT [13], alternative models were presented with various optimisations and changes, but always with the same fundamental idea of creating pretrained models that generate context-aware embeddings, and that can then be fine-tuned on a new task.

For instance, RoBERTa [15] removes the Next Sentence Prediction task, but introduces dynamic masking, meaning that the subset of masked tokens changes at every epoch. Significantly more training data is used, 160GB compared to 16GB for the original BERT model, and Byte Pair Encoding [31] (see section 3.3.1) is used instead of WordPiece [32] to tokenise the input.

There is also XLNet [16], that replaces the original MLM objective by a permutation language modelling objective, and uses Transformer-XL [37] as a building block, a new transformer architecture that solves the issue of limited sentence length from the original transformer. It was trained on 139GB of data.

There are additionally models for other languages; for French, FlauBERT [38], that uses only the MLM objective and is trained on 71 GB of data, and CamemBERT [39], based on RoBERTa that uses 138GB of data. For German, we have German BERT[8] which is simply BERT trained with 12 GB of German data. We also have models that understand multiple languages, such as multilingual-BERT,[9] a BERT model trained on multilingual data (the exact size is not documented, but the full Wikipedia dumps for the top 100 languages was used), and XLM-R [24], a RoBERTa model trained on 2.5TB of multilingual data. A list of such models can be found on huggingface's transformers website.[10]

4.2 | Improvements

4.2.1 | Classification Heads

Starting from a base transformer model, we derive multiple classifiers following some ideas from Mozafari et al. [17]. On top of the base transformer model, we connect some additional layers that we call *classification head*, and that can extract relevant information from the transformer. For all of them we suppose a sentence size of $N + 1$ (the `[CLS]` token plus N additional input tokens, including the [SEP] token and eventual padding tokens), a hidden size for the transformer of H (which is also the size of the input embeddings), and L Transformer encoder layers (the actual value of L and H depends on the specific model used, there are often *small*, *base* and *large* variants with varying sizes). Input data will be of shape $(N+1, H)$. We ignore the softmax layer at the end of all the classification heads, as it is included in the PyTorch [33] implementation of the loss function.

Base Transformer Classifier

In the first model, illustrated in figure 4.7, the output of the `[CLS]` token (of length H, situated at the beginning of the sentence) is sent into a dropout unit with probability 0.1, and a linear layer with three outputs (corresponding to our three classes; `none`, `offensive` and `hate`). By construction, the output of the `[CLS]` token should be a summarised representation of the sentence and should, therefore, contain most of the necessary information. The output with the highest value indicates the class predicted. This is the usual and default way to use a transformer model.

[8] https://deepset.ai/german-bert
[9] https://github.com/google-research/bert/blob/master/multilingual.md
[10] https://huggingface.co/transformers/pretrained_models.html

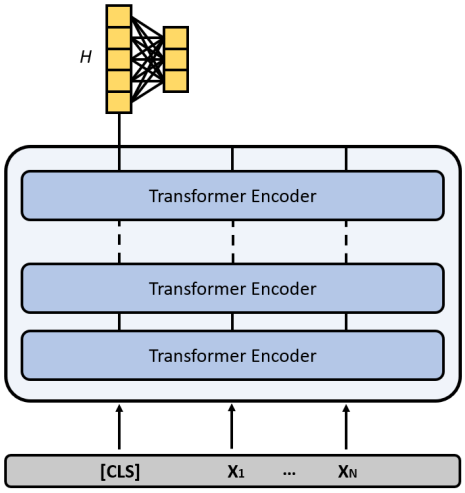

Figure 4.7. Transformer with the CLS classification head. Each transformer encoder block follows the architecture described in figure 4.4. There are L transformer encoder blocks, and we represent the first, the second and the last of them.

CLS-LSTM Classifier

In the second model, we extract the values corresponding to the `[CLS]` token at each hidden layer of the transformer. This returns L vectors of length H that are sent into a LSTM layer with a hidden size of H going from the front to the back of the network. We extract the final hidden state of the LSTM, a vector of size H, and send it into a dropout of 0.1 and a linear layer with three outputs, corresponding to our three classes (illustrated in figure 4.8).

The idea is to improve on the previous classification head by including information from the previous layers, while still only relying on the `[CLS]` token. Hopefully, using earlier iterations of the outputs of the `[CLS]` can provide more information. We can see the change of the output of the `[CLS]` token over the layers as a change over time, and a LSTM might be adapted. For instance, the earlier values might represent the sentence generally and the later values might represent it relatively to a toxicity aspect.

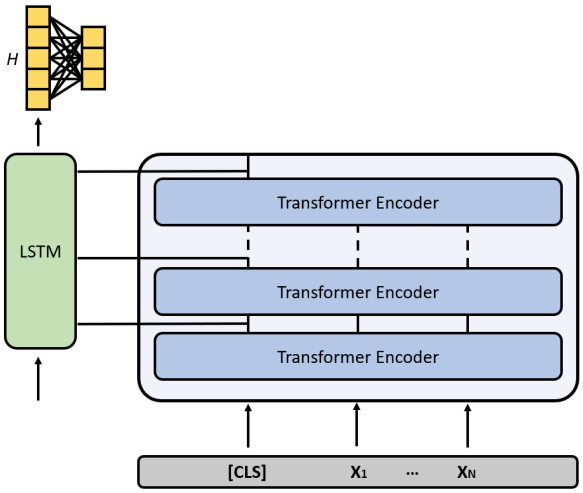

Figure 4.8. Transformer with the CLS-LSTM classification head. There are L transformer blocks with a hidden size of H, and the LSTM layer has L units with a hidden size of H.

CNN Classifier

In the third classification head, we use convolutional layers to extract information from the transformer. In the first variant, we apply L convolutional layers, one for each output of the L transformer encoder blocks. For each convolutional layer, a kernel of shape $(5, H)$ is used, and we have 200 output channels followed by a ReLU and a max-pool unit that only keeps one value per channel. This produces data of shape $(200, L)$, one vector for each layer of the transformer. We then concatenate the vectors from all layers into a vector of length $200L$, and send it into a dropout of 0.1 followed by a linear layer with three outputs. This variant is illustrated in figure 4.9.

In the second variant, we apply for each layer three different convolutional layers with kernel shapes $(3, H)$, $(4, H)$ and $(5, H)$, with 200 output channels each. As previously, a ReLU and max-pool layer are applied, and then we concatenate the data from all the layers, and for the three sizes. Similarly, we go through a dropout of 0.1 and a linear layer. We implement a similar architecture to the CNN presented in section 4.1.2, where that model is applied to each of the transformer's layers, and we then concatenate all of the outputs together.

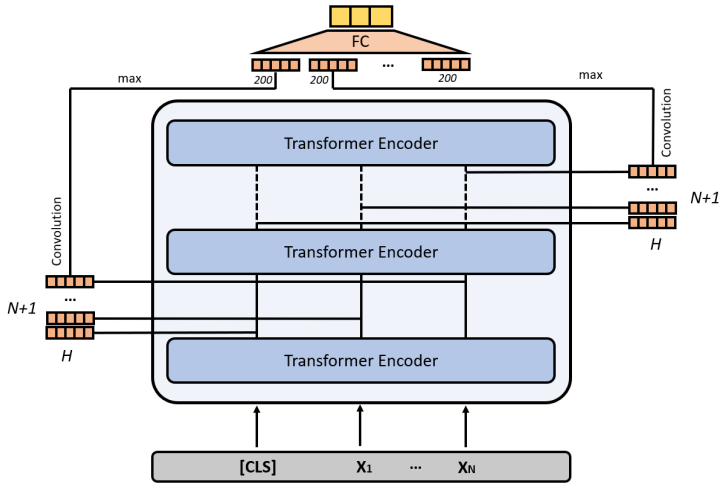

Figure 4.9. Transformer with the CNN classification head, one filter size variant, with 200 outputs channels. There are L transformer blocks, with a hidden size of H.

CNN to LSTM Classifier

For the final architecture, we extend the CNN classification head by adding a biLSTM layer. As previously, we apply a convolutional layer at each level of the transformer, either the one filter or three filter variant, and then apply the ReLU and a max-pool layer. We are left with one vector for each layer, as before. Instead of concatenating them, we send them into a bidirectional LSTM network. We then take the max-pool of all the outputs of the biLSTM, and send it to a dropout of 0.1 and a final linear layer with three outputs. Once again we have two versions, one with a single kernel of size 3 and one with three of them (sizes 3,4 and 5). The variant with a single kernel size is illustrated in figure 4.10.

The intuition is the same as with the CLS-LSTM model, there is an evolution of the hidden states when moving through the layers, and a LSTM might be a more appropriate approach than a simple concatenation to regroup data coming from different layers.

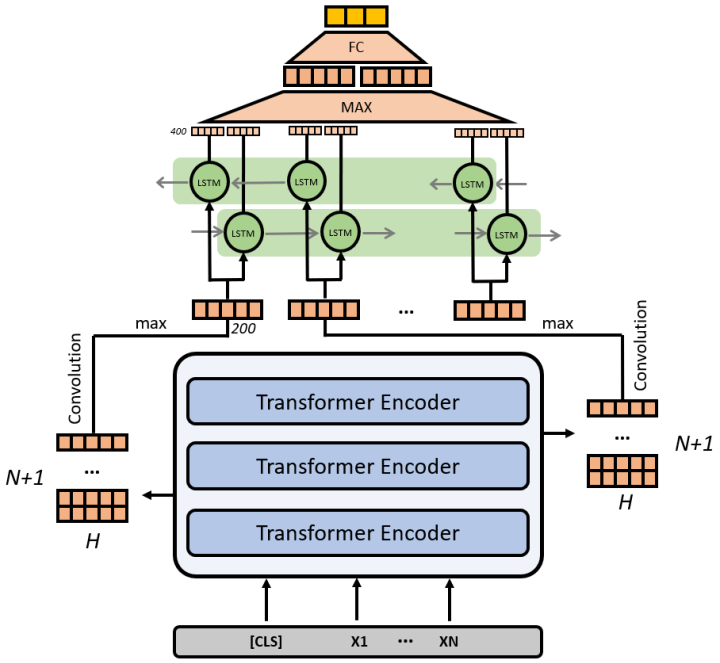

Figure 4.10. Transformer with the CNN to LSTM classification head. There are L Transformer blocks with hidden size H, the biLSTM has L units in both directions, and a hidden size of 200.

Model Complexity

Using advanced classification heads increases the complexity of the models and therefore the training time. Since the classification heads can be used with multiple models (e.g. BERT [13], RoBERTa [15], FlauBERT [38], etc.) which all have different sizes, the exact number of additional parameters depends on the actual model choice. We show in table 4.1 the relative increase in complexity depending on the classification head, to give an idea of the additional memory and training time these different architectures require.

Head	# parameters	Relative increase
None	124'645'632	+ 0%
CLS	124'699'462	+ 0.04 %
CLS-LSTM	129'444'166	+ 3.85 %
CNN	134'032'102	+ 7.53 %
mCNN	147'275'302	+ 18.16 %
CNN to LSTM	139'931'302	+ 12.26 %
mCNN to LSTM	155'296'102	+ 24.59 %

Table 4.1. Increase in the total number of parameters depending on the classification head. Values are taken from a RoBERTa-base model, and we show the number of parameters, with the increase of parameters compared to the model by itself.

It is important to remember that although the transformer is pretrained, we still have to update its parameters during training (the *fine-tuning* part). It will be the component with the largest impact on the training time. It has been shown that using a frozen transformer is detrimental to the score, although partially frozen models can reach relatively close scores compared to the normal unfrozen transformer, with lower training time [40]. We did not experiment with such optimisations and always update the full transformer.

4.2.2 | Dataset merging

When training a machine learning model, a common method to obtain significant improvement is to use more training data. In our case, we cannot increase the size of the existing datasets, but we can combine them to create a larger training set. More specifically, for a dataset of a given language, we will train a model on more data from the same language, and then evaluate it only on the original dataset. We expect this to be useful especially for datasets with a very limited amount of data (in particular in the minority classes), the assumption being that adding external data to a small dataset with weak results cannot be worse than using the original data, even if the data distributions do not exactly match. It is however important to be careful not to merge too many datasets with diverse content or themes, as the model trained might lose its compatibility with the original dataset.

4.2.3 | Multitask learning

With multitask learning, we also want to utilise information from an additional dataset to improve results, but in a more subtle manner. We split the model into two sections; the first part will be common to both datasets, and the second part will be specific to each dataset. During training, we alternate between the two datasets for each minibatch, sending the data through the common part of the model, and then through the specific part corresponding to this data. We can split the model at various levels, either having only the final linear layer specific or the complete classification head. This idea is inspired by the work of Rizoiu et al. [41], but to our knowledge it has not been applied to transformer models yet.

Figure 4.11 provides an illustration of this architecture applied on a transformer model with the CLS-LSTM classification head and with two datasets, *yellow* and *green*. We send a minibatch of *yellow* data through the transformer, and then into the *yellow* classification head, and we update all the weights on the way. The same happens with the *green* data, going through the common part of the model and then through the *green* classification head. This process is repeated for all training data. During the evaluation phase on the *yellow* data, the *green* part of the network is ignored. We can imagine an experiment where we want to use data from the Jigsaw dataset to improve results on the Davidson dataset. In this case, we would alternate between both datasets during training, but only use the classification head trained on Davidson data for the actual predictions. The transformer is better tuned for toxic content as it was also trained on the Jigsaw data, but the classification head stays specific to the Davidson data.

The concept is similar to dataset merging, in the sense that we train the model with more data, except that we make sure that at least some part of the classifier is specific to each dataset. We expect that multitask learning will work better than merging datasets in cases where the data is considerably different between the two datasets, with different class distributions. The common part will be able to learn a more general understanding of toxic language, while the specific classifier will be able to differentiate the various types of toxic content, depending on the specific definitions for each dataset involved.

In our implementation of multitask learning, the larger dataset of the two is truncated so that both datasets have the same size, but since shuffling is applied at every epoch most of the data in the larger dataset should still be used. If we had retained the original sizes, we would have been unable to alternate between the two datasets for the full epoch, and we think this could have been a problem as the common part would have started to overfit on one of the datasets, relatively to the other one. At the start of the following epoch, the classification head of the other dataset might not have supported the substantial change in outputs of the common part. For a similar reason, we initialise both classification heads with the same weights for a better connection to the common part.

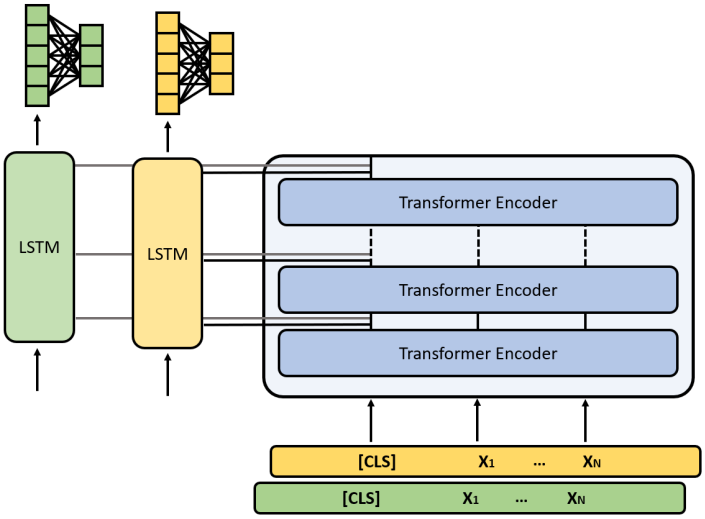

Figure 4.11. Example of a multitask architecture, here with the CLS-LSTM classification head, with data originating from two datasets, *yellow* and *green*. The transformer is common (in blue), and the classification heads are specific to the datasets (*yellow* or *green*).

4.2.4 | Further Pretraining

Instead of using the pretrained model as-is, we can do some further pretraining on it. This idea was explored in depth by Sun et al. [18], and we experiment with *within-task* pretraining. More specifically, before fine-tuning the model, we train the transformer model without the classification head for a few epochs on the original masked language model task, using our training data. With this method, the transformer model should be more familiar with the type of data that will be used to fine-tune the model, and it should help with the performance. This step is an intermediary process to facilitate the transition between the original pretrained model, and the fine-tuning, as illustrated in figure 4.12. However, there is the risk that if this process is performed for too long, the transformer will start to forget previous knowledge (from the original pretraining), and start to overfit on the new data.

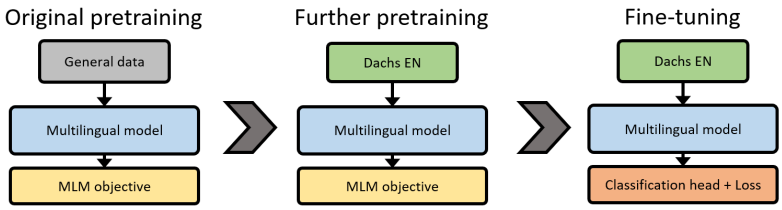

Figure 4.12. The three main steps of using a transformer model: first, the model is pretrained by its authors using general data and the MLM task, secondly, we can optionally continue training on the MLM task using target data, and finally, we fine-tune the model on target data using a classification head and a loss function.

4.2.5 | Data augmentation

One operation that can be performed to counter unbalanced classes is to augment the data in the minority classes. A naive way to achieve this is simple random oversampling, but initial experiments showed that it never works, and only makes the model overfit on this multiplicated data. A common alternative is SMOTE (Synthetic Minority Over-sampling Technique) [42] that creates new samples in minority classes by generating points in the embeddings space that are close to other samples in the same class, but it actually does not work with the transformer models as we have multiple embeddings for each input.

Instead, we try editing the copies slightly at a higher level. For instance, one could replace some words with some of their synonyms, which can hopefully bring more examples to the model while keeping correct and accurate sentences. We can also add some noise by inserting, deleting or swapping some words, but this is riskier as we might alter the meaning of a sentence and move it into another class. In particular, removing the word *"not"* might drastically alter the meaning of a sentence if the model was relying on it to make the correct classification.

We settled for synonym replacement, random insertion of synonym words and random swapping. We did not perform random deletions, as we would have taken the risk of removing offensive words that are important for the classification decision. We use the EDA[11] library [43], and we adapted it to also work with French data. We use the Open Multilingual Wordnet[12] [44] to get the synonyms, in particular we use WOLF (Wordnet Libre du Français) [45] for French and Priceton WordNet [46] for English. As far as we know, no such data were available in German, so we will not perform this operation on the German datasets. We present a few examples of sentences generated by this process in table 4.2.

Original	Generated
you should stop what you are doing	you should stop what you are practise doing
	you should stay what you are doing
just shut up and go back to your country	just shut up and run short back to your country
	just shut up and go back precisely to your country

Table 4.2. Examples of sentences before and after data augmentation.

The primary drawback of this method is the longer training time; since we augment data from the minority classes, the total amount of data after augmentation will be significantly higher. The synonyms generated are also not always accurate as the Wordnet models do not use the context of a word to find its synonyms, which can cause problems, especially with homonyms.

4.2.6 | Data cleaning and normalisation

Some simple cleaning processes can be applied to the data. Fundamental operations are removing every URL, mention and hashtag from the messages (we provide more details on the simple cleaning operations used in section 4.3.2), but it is possible to perform a more methodical cleaning. Some of the content has spelling errors, but also abbreviations that might be unknown to the transformer models. To counter this, we use MoNoise[13] [47], a text normalisation tool that was trained to clean and normalise tweets, and uses notably the Aspell spell checker,[14] word embeddings and random forests to determine the best candidates to replace words. For instance, given the sentence

> *u r rly stpid cuz u dd dat 2 all the pple jst 4 fun*

MoNoise is able recover

> *you are really stupid because you did that to all the people just for fun*

The model is available in a few languages, but unfortunately neither in French nor German, therefore we only experiment with this method on the English datasets.

[11] https://github.com/jasonwei20/eda_nlp
[12] http://compling.hss.ntu.edu.sg/omw/
[13] https://bitbucket.org/robvanderg/monoise
[14] http://aspell.net/

It does seem very good in general and is capable of correcting both spelling mistakes (*stpid* → *stupid*) and some common abbreviations (*u* → *you*, *rly* → *really*, *2* → *to*, *4* → *for*), but we were still able to find some particular cases where it does not work correctly. For instance, with a simple sentence like *"I have 2 cats"*, the *"2"* will be corrected to *"to"*. It does not always seem able to look at the context to understand the intended use of the number *"2"*. Strangely, with *"I have 4 cats"* we do not encounter the same problem, and the *"4"* is kept as-is. It would be complex to correct this behaviour, as we have both cases where we want to replace *"2"* by *"to"*, and cases where we want to keep it unchanged.

4.2.7 | Ensemble learning

Ensemble learning is a method that uses multiple models trained on the same task to obtain better performance than each model individually. The prediction of the ensemble is made by regrouping each model's prediction, for instance with majority voting (illustrated in figure 4.13). It has been shown that an ensemble between multiple instances of the same model but with random weight initialisation can improve results [48]. We can train the same model multiple times with the same data split, and due to randomness in weight initialisation and to the shuffling of data during training the resulting models will all be slightly different. Test data is given to all models and majority voting is applied to make predictions.

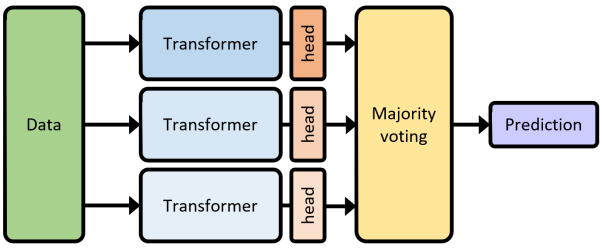

Figure 4.13. Ensemble learning with three versions of the same model and majority voting.

4.2.8 | Multilevel classification

Instead of considering our problem as a three-class problem, we can also consider it as two binary subproblems. We first train a model to differentiate `toxic` and `non-toxic` messages. Meanwhile, we train another model to differentiate `offensive` from `hate` content. We can then evaluate our model as follows: for a given message, give it to the `toxic` vs `non-toxic` classifier. If it is classified as `non-toxic`, return the class `none`. Otherwise, send it to the `hate` vs `offensive` classifier and return the predicted class (see figure 4.14).

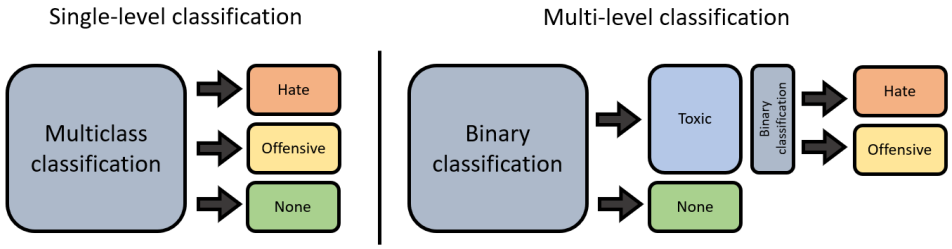

Figure 4.14. Multi-level classification compared to Single-level classification.

For this approach we use a simplified methodology; we take the architecture from the multiclass model and apply it to both classification levels. Ideally, we would need to find the optimal architecture for each level, but this would require too much time. The advantage of this architecture is that it helps with class imbalance, as `offensive` and `hate` will be regrouped into one `toxic`

class with more weight compared to `none`. It should also be easier to differentiate the subtle differences between some offensive and hateful content by having a model fully dedicated to this task. However, this model has the disadvantage of containing two sources of error, in the first and second classifier, so it may be more likely to make an error. It also requires more training time, especially for large `hate` and `offensive` classes.

4.3 | Experiments

4.3.1 | Loss function

As the loss function, we use the *Cross-Entropy Loss* (CEL). It is defined, for a multiclass problem with M classes and a true class o, as

$$\mathcal{L} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

where $y_{o,c} = 1$ if and only if the class label c is equal to the true class o, and $p_{o,c}$ is the prediction probability for the class c. Note that the CEL only takes the quality of the prediction on the true class into account, implying that a case with the correct prediction can have the same loss as a case with a wrong prediction. For instance, if we assume that class 0 in a 3-class problem is the true class, a model with predictions `[0.4, 0.5, 0.1]` will have the same cross-entropy loss as a model with outputs `[0.4, 0.3, 0.3]` (loss of $log(0.4)$ in both cases), despite having a different predicted class.

For some models, we alternatively use the *Weighted Cross-Entropy Loss* (WCEL) to assign more weight to samples originating from some classes. Since our target metric is the macro-F1 that gives the same importance to all classes, using a loss function that can increase the importance of minority classes can be beneficial. The WCEL is defined as

$$\mathcal{L}_w = -\sum_{c=1}^{M} w_c y_{o,c} \log(p_{o,c})$$

where w_c is usually inversely proportional to the number of training samples in class c. In practice, we use $w_c = \frac{1}{n_c}$, with n_c the number of samples in class c. In our experiments, we found that the WCEL is beneficial with simple models like the CNN and LSTM, giving better macro-F1 scores than with the CEL, but detrimental to the larger transformer models, giving lower macro-F1 scores compared to when using the CEL. Additional effects of this loss function are presented in section 6.3. We briefly experimented with an "*F1 loss function*" that was computed as 1 − F1-score but it was only possible to implement this for a binary problem, so we went with the CEL instead, the loss function usually selected for multiclass problems.

The PyTorch [33] implementations of the CEL and WCEL include a softmax layer, therefore it was ignored in the architecture of all our models.

4.3.2 | Data Preprocessing

For all experiments, an aggressive cleaning process was chosen. We put all content in lowercase, remove any emojis and most of the punctuation and symbols (we remove - _ / : ; " ' , * ( ) | \). We also replace a few abbreviations ("1st", "2nd", "3rd" become "first", "second" and "third", "$", "£", "%" become "dollar", "pound", "percent") and a few expressions (for instance

"*fyi*" becomes "*for your information*", "*gtfo*" becomes "*get the f*ck out*", etc.). All the hashtags, mentions and numbers are removed. We get rid of repeating words to mitigate the importance of some spammed expressions and, inside words, we only keep two occurrences of a letter appearing three or more times in a row. This should help with the various exaggerations (e.g "*it is sooooo cold*" should be considered the same as "*it is soooooooo cold*"). A few examples of data cleaning can be seen in table 4.3. We experimented with other variations of the cleaning process, where we kept and separated the hashtags into words, or replaced the mentions by special "<user>" tokens, but the difference was negligible and since it was not possible to perform a full grid search on all the variations, we adopted the original aggressive cleaning process. Moreover, removing all traces of Twitter-specific elements (hashtags and mentions) should help the models generalise better on generic text.

Original text	Cleaned text
@realDonaldTrump @SecPompeo @TuckerCarlson @FoxNews No day is complete without a racist rant from the president and Fox	no day is complete without a racist rant from the president and fox
@theJeremyVine Do they ""grate"" on you after a while?	do they grate on you after a while ?
RT @okamlord: I f*cking hate you https://t.co/DEzJoO66A2	i f*cking hate you
Amazon, vendeur de destruction massive — via @lemondefr https://t.co/r1gk7G8pHS	amazon vendeur de destruction massive

Table 4.3. Examples of sentences before and after the cleaning process. Censorship of obscene words was manually added for the report.

Examples in table 4.3 show that while the mentions might be useful in some cases (mentions to *@realDonaldTrump* or *@FoxNews* might for instance talk about controversial topics, with polarising opinions and heated debates), there are also all the other cases where the presence of such mentions does not seem to bring any meaningful information except some noise (e.g. *"@okamlord"*, *"via @lemondefr"*). We wanted a classifier based on pure textual content only and avoid bias due to the presence of some mentions. The same is true with hashtags; we do not want our models to be too reliant on them.

For the Logistic Regression classifier, CNN and biLSTM models (sections 4.1.1, 4.1.2 and 4.1.3), we use the *TweetTokenizer* from nltk[15] to separate the sentences into sequences of tokens. With the transformer models (section 4.1.4), we use each model's included tokeniser.

4.3.3 | Experimental Setup

The datasets are divided into *train*/*validation*/*test* sets with an 80%/10%/10% split ratio. We use stratified sampling to preserve the class ratio in all sets. For each experiment we train our model on the *train* set, randomly shuffled at each epoch. Four times per epoch the model is evaluated on the *validation* set and we track the evolution of the metric (macro-F1) over time. The final *test* metrics are computed using the state of the model when the *validation* metric was at its highest point. This process is repeated three times for each experiment, where the *train*/*validation*/*test* sets are randomly sampled from the data each time and we report the average and standard deviation of the macro-F1 on the test sets over the three runs. The early stopping uses the macro-F1 score instead of the loss function, as it was observed that the macro-F1 score would still increase for a while after the over-fitting syndrome started appearing on the loss function. This probably happens since the CEL and macro-F1 do not measure the same

[15] https://nltk.org/

thing: the CEL only takes the probability of the correct class into account, but does not report if the predicted class is actually correct, while the macro-F1 only measures the correctness of the predicted class. Both metrics do not measure the same aspect of the problem and thus are not synchronised; since in the end we are interested in the macro-F1 score, we preferred it for the early stopping.

We decided against using a fixed *test* set for the experiments, as its choice would have been arbitrary and would therefore not have been a reliable way of comparing models. We also decided not to use the *validation* score to compare models, as we already rely on it for the early stopping, and did not know if the difference in architectures had an impact on the variability of the metric during training (with more extreme high peaks in the curves that might impact the early stopping), and therefore on the score used to compare the models. By using the average over three random *test* sets, data that was not used during training in any way, the results should be more conclusive regarding the performance of the model.

All experiments were run with PyTorch[16] [33] on a machine running Ubuntu 18.04, with an Intel(R) Core(TM) i5-7500 quad-core CPU, 32GB of RAM, and a Nvidia GEFORCE GTX 1080 Ti with 12GB of VRAM. Training of the transformer models lasted between 15-20 minutes on the smaller models and datasets and 2-3 hours on the largest models and datasets. A large amount of VRAM was required, from 2-3 GB for the smaller models (e.g. BERT [13]) to approximately 10GB for the largest models (XLM-RoBERTa [24] or XLNet [16]). Most models exist in different sizes (with more layers and a larger hidden size), but we always use the *base* model. Larger models usually come with a small improvement in results but have significantly larger training time and memory requirements.

4.3.4 | Results

In this section, we compare the diverse architectures presented in sections 4.1 and 4.2. We start with the simple models, i.e. the Logistic Regression classifier, the CNN and the biLSTM network (sections 4.1.1, 4.1.2 and 4.1.3). We use these as a baseline to determine the usefulness of more advanced transformer models. For the transformers, we first evaluate the base models, then we look for a good combination of training improvements and architectural changes that can improve their performance. We focus on the changes and differences between architectures and methods, but not on the details of each architecture; we did not spend a lot of time tuning hyperparameters as we felt that the architectural and methodological changes were more interesting.

Simple models

More specifically, for the simple models we have:

- A Logistic Regression classifier with a bag of unigrams and bigrams, with only the 20'000 most frequent features kept (see section 4.1.1). We limit the algorithm to 2'000 iterations.
- A CNN following the architecture presented in section 4.1.2, trained with the weighted cross-entropy loss and tested with GloVe [10] and FastText [11] embeddings. The CNN has 100 output channels, kernel sizes of 3, 4 and 5 by the embedding size, and a dropout of 0.1 before the final layer.
- A biLSTM network following the architecture from section 4.1.3, trained with the weighted cross-entropy loss and tested with GloVe and FastText embeddings. The LSTM units use a hidden size of 100, and a dropout of 0.1 is applied before the final layer.

[16] http://pytorch.org

Both the CNN and biLSTM models are trained over 50 epochs, with early stopping after 2 epochs with no improvement and using the Adam optimiser [49] with a learning rate of 10^{-4} (other parameters follow the default values from the PyTorch implementation). We use a cosine scheduler that decreases the learning rate over the 50 epochs and a mini-batch size of 32. For the GloVe and FastText embeddings, we use the embedding layer from PyTorch. Since it needs to be initialised beforehand, we need to define a vocabulary; to spare memory, we do not use the full vocabulary from the trained embeddings, but only the words present in our data. Results should be identical, and it makes the model smaller. If we have unknown words in the dataset, they are added to the embeddings as random vectors.

For each model, we report the average macro-F1 score on the test set (with the standard deviation over three runs) in table 4.4. In bold, the best model, and if two models are so close that the difference is non-significant both will be marked in bold.

	Logistic	CNN		biLSTM	
Dataset	Count	GloVe	FastText	GloVe	FastText
Dachs EN	66.0 (± 0.2)	69.2 (± 0.7)	67.7 (± 0.1)	**69.7** (± 0.7)	67.1 (± 0.4)
Founta	66.4 (± 0.1)	68.9 (± 0.5)	67.4 (± 0.5)	**69.6** (± 1.0)	**70.0** (± 1.0)
Davidson	72.2 (± 1.2)	73.4 (± 0.6)	72.2 (± 1.7)	**75.9** (± 0.8)	73.9 (± 0.6)
Jigsaw	78.4 (± 0.2)	81.3 (± 0.1)	81.1 (± 0.3)	**82.0** (± 0.2)	81.8 (± 0.3)
Dachs FR	70.4 (± 0.4)	72.1 (± 0.7)	**72.6** (± 0.2)	70.2 (± 0.7)	70.7 (± 0.5)
MLMA FR	41.3 (± 2.5)	33.3 (± 8.0)	24.7 (± 1.1)	42.0 (± 3.6)	**45.0** (± 2.3)
Dachs DE	44.2 (± 0.5)	43.1 (± 1.4)	38.7 (± 6.4)	44.0 (± 1.2)	**44.7** (± 0.4)
Germeval	50.0 (± 0.4)	44.6 (± 11.2)	47.2 (± 17.2)	51.0 (± 0.5)	**55.4** (± 0.7)

Table 4.4. Results for the simple classifiers. We report the average test score over three runs with the standard deviation.

The first conclusion that can be drawn from these results is that the more complex models often perform better than the simple Logistic regression model and that the biLSTM usually outperforms the CNN, probably due to its more complex architecture, most notably its ability to see long-term dependencies better as it is not limited to a maximum of five consecutive tokens like the CNN. Secondly, in English GloVe embeddings are often better than FastText embeddings, but in French and German the FastText embeddings are typically better. There are a few exceptions, in particular Dachs FR performs better with the CNN, and with MLMA FR and Dachs DE the GloVe embeddings perform better than FastText when using the CNN. However, we see in a few instances very large standard deviations; this happens when the model has a lot of difficulties during training, and in one or two of the trials it does not learn anything at all and stays in its initial state, a good indication that the model is not adapted to the dataset.

We see that the results in English are generally better than the results in French and German. This, of course, depends on the quality of the datasets, but except for Dachs FR the results for French and German are really bad. For the MLMA dataset, this can be explained by the very small size of the dataset, and for Dachs DE the low amount of `hate` data (a little over 2.5% of the total) certainly does not help. More generally for the German datasets, one can also hypothesise that the complexity of German grammar might be at cause; there are often long and complex dependencies in German, and the various word declinations (dative, accusative, genitive, etc.) might add too many variations of the same word, and these models might not be smart enough to detect those variations as the same word. A more careful data prepossessing process with lemmatisation and morphological analysis could help.

For a baseline, it is also important to have more information about the scores on the different classes. For each dataset, we report in table 4.5 the macro-F1, and the recall and precision on all classes for the best simple models according to table 4.4. This gives us a more precise baseline, before moving on to the transformer models.

			None		Offensive		Hate	
Dataset	Model	macro-F1	P	R	P	R	P	R
Dachs EN	biLSTM+GloVe	69.7	83.7	87.7	66.4	60.3	64.4	56.9
Founta	biLSTM+GloVe	70.0	95.2	96.0	83.5	88.4	37.9	22.9
Davidson	biLSTM+GloVe	75.9	80.0	91.3	94.9	93.3	53.9	44.0
Jigsaw	biLSTM+GloVe	82.0	87.4	91.1	88.8	83.5	69.8	71.5
Dachs FR	CNN+FastText	72.6	86.9	92.3	69.4	59.6	74.5	56.3
MLMA FR	biLSTM+FastText	45.0	51.0	41.8	68.9	68.3	27.7	26.9
Dachs DE	biLSTM+FastText	44.7	91.1	94.9	35.5	20.8	15.9	14.1
Germeval	biLSTM+FastText	55.4	81.5	86.0	36.8	34.1	52.2	43.1

Table 4.5. Detailed results for the best simple classifiers (according to table 4.4).

We can see that with some datasets there is a significant imbalance between the different classes. With Founta, Davidson and MLMA FR, the metrics on `hate` are very much lower than for the `offensive` class. This phenomenon is also present on other datasets, but to a lesser extent, for instance with Jigsaw. This phenomenon is expected, as the `hate` class is usually by far the class with the least data. It still needs to be seen if the transformer models can correct this issue.

Base Transformer models

We now present results for the transformer models (section 4.1.4). We first evaluate existing models taken directly from hugginface's transformers library,[17] and for the best model for each dataset we try the different classification heads (section 4.2.1) and data combination methods (sections 4.2.2 and 4.2.3). Afterwards, we evaluate the different training improvements presented in sections 4.2.4 to 4.2.8, each of them independently, and apply the ones that are useful to obtain our final models.

We train all models for 10 epochs with a cosine scheduler and the Adam optimiser [49], with early stopping on the macro-F1 after two epochs without improvement, as the model usually reaches a maximal score after three to five epochs. The learning rate for the transformer part is 2×10^{-5} on the recommendation of Devlin et al. [13], and the learning rate for the classification head is 5×10^{-5} (we found during experimentation that a higher rate for the classification head was beneficial). A mini-batch size of 64 was used for all models.

In table 4.6 we report for each dataset the scores for a few transformer models. Those models are taken as-is, without any change to the classification head (we use the default CLS classification head). We always take the *base* models, and the *cased* instead of *uncased* version if applicable.

Firstly, we see that BERT [13], the original transformer model, underperforms compared to the newer RoBERTa [15] model. This difference is highly dependent on the dataset, with only a 0.5% difference with RoBERTa on Dachs EN and more than 8% on Jigsaw, but RoBERTa is always the best model for English data.

[17] https://huggingface.co/transformers/pretrained_models.html

	Transformer Model			
Dataset	BERT [13]	RoBERTa [15]	XLNet [16]	XLM-RoBERTa [24]
Dachs EN	74.5 (± 1.6)	**74.9** (± 0.1)	73.4 (± 0.6)	74.6 (± 0.6)
Founta	73.0 (± 1.1)	**73.8** (± 0.7)	73.1 (± 1.5)	73.2 (± 0.5)
Davidson	75.7 (± 0.7)	**77.8** (± 1.3)	74.2 (± 0.9)	77.1 (± 1.5)
Jigsaw	75.4 (± 0.1)	**83.7** (± 0.4)	81.9 (± 0.6)	82.0 (± 0.6)
		FlauBERT [38]	CamenBERT [39]	XLM-RoBERTa
Dachs FR		**78.7** (± 1.5)	76.7 (± 1.1)	75.6 (± 1.1)
MLMA FR		**46.5** (± 1.3)	44.3 (± 0.9)	42.5 (± 2.9)
			german-BERT[18]	XLM-RoBERTa
Dachs DE			52.1 (± 0.8)	**52.7** (± 2.8)
Germeval			60.2 (± 1.2)	**61.6** (± 2.3)

Table 4.6. A few different transformer models, with the CLS classification head.

Furthermore, in English and French the best monolingual models always outperform the multilingual model. This is expected, as RoBERTa, FlauBERT [38] and XLM-RoBERTa [24] are very similar architecturally, but monolingual models are more adapted since they do not include all unnecessary data for other languages. In the case of the German datasets, the multilingual model here is better, presumably due to the larger amount of training data (2.5TB in total for XLM-R, with 66GB of German data) compared to german-BERT (only 12GB). We can already conclude that if we want to use those multilingual models in the future to improve our results with cross-lingual toxicity detection, we will have a disadvantage that will need to be compensated before hoping to surpass the original monolingual score. These results give us a baseline for the transformer models, and they show what kind of results we can hope for when using the pretrained models from the huggingface's transformers library directly, without any changes.

Classification heads

In table 4.7 we show for each dataset, and using the best transformer models from table 4.6, the scores using the different classifications heads introduced in section 4.2.1.

		Classification Head					
Dataset	Model	CLS	CLS-LSTM	CNN	mCNN	C2L	mC2L
Dachs EN	RoBERTa	74.9 (± 0.1)	74.9 (± 0.2)	**75.1** (± 1.1)	74.3 (± 1.0)	74.7 (± 1.0)	74.8 (± 0.7)
Founta	RoBERTa	**73.8** (± 0.7)	73.3 (± 1.0)	**73.5** (± 1.2)	72.8 (± 1.1)	72.0 (± 1.4)	73.0 (± 0.6)
Davidson	RoBERTa	**77.8** (± 1.3)	75.9 (± 0.1)	75.8 (± 0.5)	76.8 (± 0.5)	**78.1** (± 1.0)	77.5 (± 0.6)
Jigsaw	RoBERTa	83.7 (± 0.4)	83.6 (± 0.8)	83.6 (± 0.4)	83.2 (± 0.4)	83.2 (± 0.4)	**84.1** (± 0.6)
Dachs FR	FlauBERT	78.7 (± 1.5)	78.4 (± 2.8)	**79.6** (± 0.3)	78.6 (± 1.1)	77.9 (± 1.2)	78.0 (± 1.7)
MLMA FR	FlauBERT	**46.5** (± 1.3)	43.6 (± 0.9)	**46.9** (± 1.8)	**46.3** (± 4.6)	30.0 (± 3.3)	29.0 (± 2.3)
Dachs DE	XLM-R	**52.7** (± 2.8)	50.1 (± 6.4)	51.2 (± 0.7)	50.4 (± 0.6)	38.3 (± 13.8)	39.3 (± 22.1)
Germeval	XLM-R	**61.6** (± 2.3)	60.0 (± 1.3)	**61.5** (± 3.0)	61.3 (± 1.1)	60.2 (± 1.3)	**62.6** (± 1.5)

Table 4.7. Results for the different classification heads. CLS is the default model. CLS-LSTM uses a LSTM on all hidden states of the `[CLS]` token. CNN and mCNN are the 1-filter and 3-filters variants of the CNN classification head. C2L and mC2L are the 1-filter and 3-filters variants of the CNN to LSTM classification head.

The first thing we observe is the similarity of the scores for the different classification heads, with close averages and relatively large standard deviations. Since we use cross-validation and randomly sample the *train/validation/test* sets, we have different data for each run, which creates some variance in the results. Besides, there is some inherent variance during the training of a transformer model; the initialisation of weights has an non-negligible impact [48], so even if we retained the same data in all trials we would get some variation.

Secondly, although the best model is not the same for all datasets, CLS and CNN are often among the better classification heads. For each of these heads, there are a lot of optimisations that could be done; we could add some layers, some regularisation, change kernel sizes, change the learning rate, etc., so it might be possible to achieve different and better results. The results here show that there is some potential and that the way information is extracted from the transformer has some importance, even if the improvement here is not as large as hoped. According to results from Mozafari et al. [17], this improvement seems to be highly dependant on the dataset and classification task. On one of their datasets, they achieve significant improvement by using a specialised classification head, while the difference is negligible on another dataset.

Finally, we note a few instances with very low scores, these happen when in one or more of the three trials the model is not able to learn meaningful information. It sometimes gets stuck in a state close to the initial random state, giving a score of around 33%, sometimes lower, equivalent to a random prediction. It only happens with harder datasets, here MLMA FR and Dachs DE. The large standard deviation in the case of Dachs DE shows that the model was still able to fit to the data at least one time. We considered ignoring those very bad trials but decided against as it is a valuable indication of the quality of the model and its compatibility with a dataset.

Overall, these results show that the most important part of the architecture is still the transformer itself and not the way the data is extracted. Since all classification heads also have access to the output of the `[CLS]` token, which is supposed to give all necessary information regarding the input, the additional information from the more complex classification heads might be redundant in some cases.

Dataset Combining

We now try to combine data from multiple datasets. This can be done either by merging directly all the datasets of the language (section 4.2.2) or by using multitask learning (two versions possible as described in section 4.2.3, with either the final layer or the classification head specific to each dataset). We try combining each dataset *(primary* dataset) with a new dataset (*secondary* dataset) created from all remaining data in the language (e.g. with Dachs EN, we combine it with the combination of Davidson, Founta and Jigsaw data). Better results might be achievable by testing other subsets of data for the secondary datasets, but we did not have time to consider all possible combinations. The goal here was to put as much diversity as possible in the training set to hopefully better fit the test data.

The different methods can be seen as different levels of data combination, and we can place them on a scale; on one side, we have a model completely specific to the actual dataset, with no combining (*None*). Then, when we do multitasking at the classification head level (*MT-class*), the model still has quite a large part that is specific to the dataset. With the multitasking only at the final layer (*MT-final*), the model has less specific parts, only the final layer, and most of it is common. Finally, with full dataset merging (*Merge*), none of the model is specific to the dataset. We show those four approaches in table 4.8. Results in the *None* column are taken from table 4.7.

		Combining type			
Dataset	Model	None	MT-class	MT-final	Merge
Dachs EN	RoBERTa+CNN	**75.1** (± 1.1)	**75.3** (± 0.6)	**75.3** (± 1.0)	70.5 (± 1.4)
Founta	RoBERTa+CLS	**73.8** (± 0.7)	**74.2** (± 1.0)	**73.9** (± 0.9)	69.6 (± 0.5)
Davidson	RoBERTa+C2L	**78.1** (± 1.0)	75.3 (± 0.2)	76.5 (± 1.6)	75.7 (± 0.4)
Jigsaw	RoBERTa+mC2L	**84.1** (± 0.6)	83.0 (± 0.4)	83.4 (± 0.3)	81.7 (± 0.6)
Dachs FR	FlauBERT+CNN	**79.6** (± 0.3)	74.2 (± 1.6)	74.7 (± 1.3)	77.0 (± 0.9)
MLMA FR	FlauBERT+CNN	46.9 (± 1.8)	46.7 (± 2.5)	46.4 (± 1.4)	**48.1** (± 1.5)
Dachs DE	XLM-R+CLS	**52.7** (± 2.8)	51.6 (± 2.1)	**52.5** (± 3.6)	48.6 (± 1.0)
Germeval	XLM-R+mC2L	**62.6** (± 1.5)	61.2 (± 0.6)	60.5 (± 1.0)	55.4 (± 0.8)

Table 4.8. Effects of different levels of data combination. Each dataset is combined with the other datasets of the same languages, either by merging or by multitask learning. For multitask learning, we have two variants: *MT-class* has the whole classification head specific to each dataset, *MT-final* only has the final layer.

The performance of data combination is highly dependent on the datasets themselves, but in most cases combining data is detrimental to the score. It might be slightly useful for the datasets with the lowest scores in each language (Dachs EN, Founta, MLMA FR and Germeval) since we include data from better datasets, but it is detrimental to datasets with better scores (as we add data from datasets that do not perform as well). Most of the time we are within a margin of error, meaning that combining data is not very useful especially considering the added training time. The only clear improvement is with data merging for MLMA, where we gain a few percentage points, but the score still stays very low.

When combining datasets, multitasking is usually a better approach compared to naive merging, except for MLMA FR and Dachs FR. This is due to the disparity in sizes of the two datasets, and the fact that we reduce the largest dataset to the size of the smallest dataset with multitask learning. The Dachs FR dataset will be significantly reduced when combined with the smaller MLMA FR dataset, leading to a smaller total amount than with Dachs FR only. For MLMA FR, adding the Dachs FR data with multitask will only double the total size, while merging the two datasets will result in a remarkably larger total dataset, which is extremely valuable.

Ultimately, the fundamental problem of data combination is the domain gap between the datasets. They do not all have the same exact definitions and data distribution, and the model will have to generalise more to fit all regrouped datasets, therefore the score on a single dataset will be worse. Dataset combining might still be useful in a zero-shot approach where the evaluation dataset is not used during training; if we used multiple datasets during training, there should be a higher chance of generalising correctly on unseen evaluation data. We did not verify this claim due to time constraints.

All these architectural changes (*transformer type*, *classification head* and *data combining*) have some non-negligible influence. We cannot conclude that a method is always effective, as the results vary significantly between datasets and there is a lot of variance. However, it is apparent from the results in tables 4.6, 4.7 and 4.8 that it is possible to improve results from the original model by combining some of those options. This gives us a new baseline, summed up in table 4.9, from which we can then try the remaining training improvements.

Dataset	Model	Score
Dachs EN	RoBERTa + CNN + MT-class	75.3 (± 0.6)
Founta	RoBERTa + CLS + MT-class	74.2 (± 1.0)
Davidson	RoBERTa + C2L	78.1 (± 1.0)
Jigsaw	RoBERTa + mC2L	84.1 (± 0.6)
Dachs FR	FlauBERT + CNN	79.6 (± 0.3)
MLMA FR	FlauBERT + CNN + Merge	48.1 (± 1.5)
Dachs DE	XLM-R + CLS	52.7 (± 2.8)
Germeval	XLM-R + mC2L	62.6 (± 1.5)

Table 4.9. Baselines when selecting the best transformer model, with the best classification head and merging technique.

Further training improvements

We now evaluate the effects of each of the remaining training improvements individually, then combine the options that were beneficial to create a final model. More specifically, we consider the following methods, with results reported in table 4.10:

- *Pretraining* (section 4.2.4): pretrain the transformer model on the training data.[19] This pretrained model is then used as a starting point for the usual classification. In the cases of data combining, we use all combined data for the pretraining.
- *Data augmentation* (section 4.2.5): augment the minority classes to reach a similar size to the majority class (for a class C, and a majority class M, augment by a factor $\left\lfloor \frac{|M|}{|C|} \right\rfloor$).
- *Data Normalisation* (section 4.2.6): use MoNoise [47] to normalise the data on top of the existing cleaning process. Perform synonym replacement, random insertions and random swaps, all with probability 0.3.
- *Ensemble* (section 4.2.7): make a majority vote between three versions of the same model.
- *Multilevel classification* (section 4.2.8): train a model to differentiate toxic and non-toxic content, and train another model to differentiate hateful and offensive content.

		Options				
Dataset	Base	Pretraining	Augmentation	Normalisation	Ensemble	Multilevel
Dachs EN	75.3 (± 0.6)	74.9 (± 0.3)	73.6 (± 0.9)	74.8 (± 0.2)	76.1 (± 0.5)	75.0 (± 0.9)
Founta	74.2 (± 1.0)	73.5 (± 0.6)	71.4 (± 1.7)	73.8 (± 0.9)	73.7 (± 1.1)	70.9 (± 0.7)
Davidson	78.1 (± 1.0)	76.3 (± 1.1)	76.0 (± 0.9)	75.7 (± 0.4)	77.4 (± 1.8)	77.2 (± 1.1)
Jigsaw	84.1 (± 0.6)	83.2 (± 1.8)	83.7 (± 0.2)	83.7 (± 0.5)	84.2 (± 0.0)	82.7 (± 0.5)
Dachs FR	79.6 (± 0.3)	78.7 (± 1.5)	77.0 (± 1.3)	-	78.2 (± 0.9)	77.2 (± 0.6)
MLMA FR	48.1 (± 1.5)	45.5 (± 3.3)	41.3 (± 3.3)	-	46.9 (± 2.3)	43.5 (± 2.5)
Dachs DE	52.7 (± 2.8)	51.4 (± 1.3)	-	-	49.8 (± 2.9)	51.4 (± 5.9)
Germeval	62.6 (± 1.5)	62.4 (± 1.7)	-	-	62.6 (± 1.9)	58.6 (± 2.5)

Table 4.10. Effects of additional training options, evaluated individually. A dash (-) means that the improvement could not be tested (e.g. missing resources).

[19] We use an utility from huggingface's transformers: https://github.com/huggingface/transformers/tree/master/examples/language-modeling

Unfortunately, it seems that the vast majority of these options do not work. The standard deviation shows that there is not a large difference between the options, thus we cannot conclude with confidence that they never work, but it also shows that there is not a lot of potential. We can think of a few reasons why these methods are ineffective:

- For *pretraining*, the problem might come from the pretraining duration. If we pretrain the model for too long, the transformer might start to overfit on the training data and will then encounter some difficulties with the test data. No option to reduce the training time was found in the used script. We could maybe reduce the size of the data, but the model might start to overfit even more. Another solution could be to pretrain on significantly more data to avoid this overfit.

- For *augmentation*, we think that too much noise is added. The WordNet models do not use context to generate synonyms, so a lot of them might be inaccurate and alter the meaning of the sentence.

- For *data normalisation*, we think that some of the incorrectly spelt words have some use for the classification and that correcting them can have a negative impact. There is perhaps a correlation between the level of grammatical and orthographical correctness of a message and its toxicity, and such precious information is lost with normalisation.

- For *ensemble learning*, the problem might come from the distinction between the test and validation sets. The three models for the ensemble are selected to maximise the validation score but do not consider the test score. When ensembling three models optimised for the validation set, we might start to overfit in some way on this validation set, and the ensemble will be less capable of generalising with other data, here the test set.

- For *multilevel classification*, a source of error might be the grouping of `hate` and `offensive` content. The first level of the classification (binary classification `toxic` or not) might struggle to find common patterns for the `hate` and `offensive` classes, as `offensive` content might be more similar to `none` than `hate` in some cases, leading to misclassifications.

Summary of results

We summarise in table 4.11 the architecture and options chosen for each dataset. We report in table 4.12 more detailed results for the models, where we compare the best simple model (among Logistic Regression, CNN and biLSTM), with the base transformer model (i.e. a transformer model with the CLS classification head and no other improvement, see models in table 4.6), and the improved transformer model (a model with all options enabled, see table 4.11).

Dataset	Model	Head	Combining	Pretrain	Aug	Norm	Ensemble	Multilevel	Final score
Dachs EN	RoBERTa	CNN	MT-class	✗	✗	✗	✓	✗	76.1 (± 0.5)
Founta	RoBERTa	CLS	MT-class	✗	✗	✗	✗	✗	74.2 (± 1.0)
Davidson	RoBERTa	C2L	None	✗	✗	✗	✗	✗	78.1 (± 1.0)
Jigsaw	RoBERTa	mC2L	None	✗	✗	✗	✓	✗	84.2 (± 0.0)
Dachs FR	FlauBERT	CNN	None	✗	✗	-	✗	✗	79.6 (± 0.3)
MLMA FR	FlauBERT	CNN	Merge	✗	✗	-	✗	✗	48.1 (± 1.5)
Dachs DE	XLM-R	CLS	None	✗	-	-	✗	✗	52.7 (± 2.8)
Germeval	XLM-R	mC2L	None	✗	-	-	✗	✗	62.6 (± 1.5)

Table 4.11. Summary of the architectural choices and other options used for each dataset. We show options tested and kept (✓), options tested but dismissed (✗), options not tested (-) and the macro-F1 score obtained when training a model with all options kept enabled.

			None		Offensive		Hate	
Dataset	Model	m-F1	P	R	P	R	P	R
Dachs EN	Simple	69.7	83.7	87.7	66.4	60.3	64.4	56.9
	Transformer	74.9	86.6	**88.7**	**70.5**	**67.3**	72.6	64.8
	Transformer+	**76.1**	**86.8**	**88.7**	70.1	66.9	**74.7**	**69.9**
Founta	Simple	70.0	95.2	96.0	83.5	88.4	37.9	22.9
	Transformer	73.8	96.1	**96.3**	**84.7**	88.1	45.0	34.8
	Transformer+	**74.2**	**96.4**	95.8	83.7	**89.2**	**46.8**	**36.7**
Davidson	Simple	75.9	80.0	91.3	94.9	93.3	53.9	44.0
	Transformer	77.8	87.2	**91.5**	**95.3**	93.7	48.0	51.3
	Transformer+	**78.1**	**88.9**	90.3	**95.3**	**94.3**	**48.7**	**52.0**
Jigsaw	Simple	82.0	87.4	**91.1**	**88.8**	83.5	69.8	71.5
	Transformer	83.7	88.4	90.4	88.1	84.9	73.4	**77.7**
	Transformer+	**84.2**	**88.7**	90.8	88.2	**85.6**	**76.8**	75.2
Dachs FR	Simple	72.6	86.9	**92.3**	69.4	59.6	74.5	56.3
	Transformer	78.7	**91.0**	89.9	68.7	**71.9**	77.4	73.7
	Transformer+	79.6	90.7	90.5	**69.9**	71.2	**80.8**	**74.9**
MLMA FR	Simple	45.0	**51.0**	41.8	68.9	68.3	27.7	**26.9**
	Transformer	46.5	41.4	**48.8**	67.5	65.9	32.1	26.0
	Transformer+	**48.1**	41.2	46.8	**69.4**	**72.6**	**36.5**	24.7
Dachs DE	Simple	44.7	91.1	94.9	35.5	20.8	15.9	14.1
	Transformer	**52.7**	**92.7**	**95.1**	**44.2**	**35.2**	**37.7**	**22.6**
	Transformer+	**52.7**	**92.7**	**95.1**	**44.2**	**35.2**	**37.7**	**22.6**
Germeval	Simple	55.4	81.5	86.0	36.8	34.1	52.2	43.1
	Transformer	61.6	85.2	84.7	47.5	**46.0**	52.1	54.4
	Transformer+	**62.6**	**85.5**	**87.7**	**53.6**	43.1	**53.9**	**55.1**

Table 4.12. Summary of the scores for each dataset. *Simple* shows the scores of the best simple model from table 4.4. *Transformer* shows the results when using a transformer model directly from the *transformers* library (see table 4.6), and *Transformer+* shows the results when training models with all good options enabled, as detailed in table 4.11.

It is clear that the transformer models outperform the traditional biLSTM and CNN based methods, often by a significant margin. Besides, in the cases where the improvement on the macro-F1 is not very big, we usually see one metric where the improvement is considerable. For instance on Jigsaw, with only 1.7% of improvement on the macro-F1, we have more than 6% of improvement for the recall on `hate`, an important metric for this task. Similar behaviours can be observed with the Founta and Davidson datasets. With the improved transformer models, the results are slightly higher than with the base transformer model. Not all datasets are similarly affected, but once again we have some cases where specific metrics are particularly improved. On Dachs EN, we gain a lot of recall on `hate` and only lose a small amount on the `offensive` metrics. Similar behaviours were observed with the Founta and Davidson datasets. With French and German data, we also improve the scores from the simple model to the base transformer, and then to the improved transformer (Dachs DE being the exception, no option improved the results for the transformer model).

For the most part, the improvements to the transformer models are not extremely useful. In most cases, they provide better scores on some metrics, but it is relatively unpredictable and we cannot significantly improve the scores with these methods. The largest improvement comes from the use of transformer models in the first place.

In figure 4.15 we plot the evolution of metrics between the simple models and advanced transformer models (values taken from table 4.12).

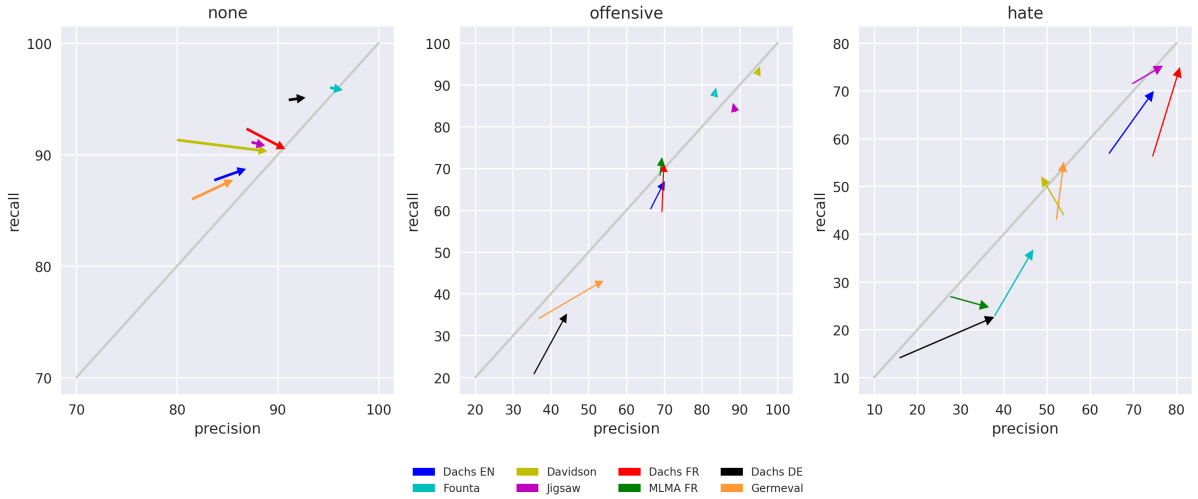

Figure 4.15. Evolution of metrics from the simple models to the improved transformers for the `none`, `offensive` and `hate` classes. The tail of the arrow shows the precision and recall of the simple model, the head shows the metrics for the improved transformer.

We observe a clear trend of improvement for all models, as most arrows tend to move towards the top right corner (representing the perfect model). More interestingly, the arrows often point towards the grey line representing the equilibrium between precision and recall, which corresponds to an improvement of recall on the `offensive` and `hate` classes, and an improvement of precision on the `none` class.

Finally, we report in table 4.13 results obtained when considering a binary classification problem. We use for each dataset the best model for the multiclass problem (table 4.11), but we train it with only two classes, `toxic` if the label was `hate` or `offensive`, and `none` otherwise.

We observe a significant increase in the scores with Founta and Davidson; these two datasets had problems differentiating `hate` and `offensive`, but when regrouping the two classes this problem disappears and we have very good results. While we used to have better results for the recall of `hate` and `offensive` with Dachs FR than with MLMA FR, now that we have regrouped the two classes MLMA FR gets better recall on `toxic` than Dachs FR. The score on the `none` class is still as bad, but this also shows that one problem of MLMA FR was the differentiation of `hate` and `offensive`.

The problem of differentiating the `hate` and `offensive` classes does not only come from the models, but also the datasets. For some of them, we observed some inconsistencies in the annotations of content in the `offensive` and `hate` classes. It was observed that the presence of some keywords would make the annotator immediately label the tweet as `hate`, without considering the context the word is used in [2], which can lead to similar content being in both classes if there is too much bias with some annotators.

		None		Toxic	
Dataset	m-F1	P	R	P	R
Dachs EN	79.3	86.4	88.0	72.8	69.9
Founta	92.4	95.7	96.7	89.9	87.2
Davidson	93.7	88.1	91.0	98.2	97.5
Jigsaw	88.3	87.7	91.1	89.2	85.2
Dachs FR	84.4	90.3	92.6	80.2	74.7
MLMA FR	62.7	45.4	42.4	81.2	82.6
Dachs DE	67.5	92.2	95.9	51.0	34.3
Germeval	74.8	85.0	83.7	64.2	66.4

Table 4.13. Metrics for the binary classification task (`toxic` or `none`), using the best architectures from the multiclass problem.

Globally, we get acceptable results on the `toxic` class, and in the end it is the most important thing. For most applications, we first want to detect all toxic content (i.e. maximise the recall on `toxic`) and not let anything through. Differentiating `hate` and `offensive` is less important in most cases. Naturally, one could imagine an application where offensive content is allowed but not hateful content, in which case the multiclass approach is useful. One could also adapt the model into one that gives a toxicity score, from 0 if offensive only to 1 if hateful, we would then be able to easily tune the system for the desired application. We opted for the multiclass approach as it gave more understandable results (with a multiclass problem, the recall and precision metrics are directly mapped to real-world performance on a moderation system), but also since all annotations in the datasets and previous works already follow the multiclass approach.

4.3.5 | Comparison with previous results

It is important to compare our results with previously published results on the same datasets to know if our models are working correctly. Unfortunately, it is often hard to find studies that use exactly the same data, processing, classification task and similar models to allow for precise comparison. Sometimes some classes are ignored, sometimes datasets are merged, and we do not always have all the original data at our disposal.

We were still able to find two datasets where we had a similar enough classification task to compare: the Dachs and Davidson datasets. For the others, we could not find adequate experiments; there were no experiments exclusively on Founta (it was merged with other datasets [20]), MLMA FR had experiments with more classes and mixed with the other languages of that dataset [3] (English and Arabic) and Jigsaw and Germeval were used for machine learning competitions, again with different goals, so we could not compare with them either.

Dachs dataset

In the original Dachs paper [1], the task differs from ours; they perform two binary classification tasks, `hate` or `not-hate` and `attack` or `not-attack`. They tested a few models, but not the transformers, so we would like to see how these perform compared to their models. We unfortunately do not have all the data that was used in the original experiment; table 4.14 reports the amount of data we have for the Dachs dataset, with the percentage of the original amount we were able to collect. We still hope to improve their results as we have more advanced architectures.

	Hate speech		Personal Attack	
lang	positive	negative	positive	negative
en	3'906 (54%)	57'916 (68%)	18'001 (63%)	43'811 (70%)
fr	1'716 (63%)	30'555 (84%)	8'798 (72%)	23'473 (87%)
de	779 (46%)	29'496 (70%)	2'642 (57%)	27'650 (71%)

Table 4.14. Data for the original two sub-problems in the Dachs datasets.

For all languages, we use models with the best architectural choices and options, except for self-ensemble and data combining, as we do not have a secondary dataset with the same annotations to use (see table 4.11), and compare the score with their best non-ensemble model.

For the two classification tasks, Charitidis et al. [1] reported the macro-precision and macro-recall (the averages of the precision or recall on the two classes), the macro-F1 and toxic-F1 (i.e. the F1 score on the `hate` class for the `hate` problem and the F1 score on the `attack` class for the `attack` problem). Since it is unclear if they used some type of average on their results, we use our usual method with the average on the test set over three runs. We report our results alongside the results provided by Charitidis et al. [1] in table 4.15.

		Hate		Attack	
lang	metric	[1]	ours	[1]	ours
en	m-P	83	**87**	78	**79**
	m-R	78	**82**	**79**	**79**
	m-F1	80	**85**	78	**79**
	t-F1	64	**71**	**71**	**71**
fr	m-P	81	**91**	81	**83**
	m-R	82	**84**	82	**83**
	m-F1	82	**87**	81	**83**
	t-F1	66	**75**	**75**	**75**
de	m-P	**67**	62	**70**	**70**
	m-R	**71**	58	**73**	68
	m-F1	**69**	59	**71**	69
	t-F1	**40**	20	**49**	43

Table 4.15. Comparison with results from Charitidis et al. [1] on the Dachs datasets. We use transformer models with custom classification heads (en: RoBERTa + CNN, fr: FlauBERT + CNN, de: XLM-R + CLS). They use a CNN-based model.

To begin with, it is important to note that the models used here were optimised for the multiclass problem (`hate`, `offensive` and `none`) and not on the two binary classification problems presented here. Repeating the same process as previously (evaluating all options to determine the best combination) but directly on the target tasks might give better results. Still, we see that in English and French we are able to get better results than the authors of the dataset. We have more complex models, but also clearly less data. This is a good sign that transformer models are indeed useful and better than the simpler models. In German, we are not able to match their scores, probably due to the very limited amount of data at our disposal, especially on the `hate` class. On the `attack` class, with a larger percentage of the original data, we are closer although we still do not match their results.

Davidson dataset

Here, we compare our models to those of Mozafari et al. [17] on the Davidson dataset. In their paper the weighted F1-score is used as a metric, which by itself does not give a lot of information especially about the minority classes, but they also report a confusion matrix which allows us to recompute all other metrics for a better comparison. The metrics given in table 4.16 are for their best model, BERT [13] using a custom classification head based on a CNN, and we compare those with our best model for this dataset, RoBERTa [15] with the CNN to LSTM classification head (see section 4.2.1). Once again, since we do not know their methodology for reporting results, we report the average over three runs on random test sets for our models.

		P	R	F1
[17]	none	91.6	92.2	91.9
	offensive	94.0	97.3	95.6
	hate	59.2	29.5	39.4
	macro avg	81.6	73.0	75.6
	weighted avg	91.6	92.6	**91.6**
Ours	none	88.9	90.3	89.6
	offensive	95.3	94.3	94.8
	hate	48.7	52.0	50.0
	macro avg	77.6	78.9	**78.1**
	weighted avg	91.5	91.2	91.3

Table 4.16. Comparison on the Davidson dataset. They use BERT with a CNN-based classification head, we use RoBERTa with the CNN to LSTM classification head.

If we look at the main metric they report, the weighted F1-score, we do not quite match their results. However, our model has a better macro-F1 score. The `hate` class performs significantly better with our model, and this has a larger influence on the macro-F1 score compared to the weighted-F1 score. We score a little lower with the `none` and `offensive` classes, but we think the gain in recall on `hate` makes it worth.

5 | Cross-lingual Toxicity Detection

In this chapter, we present our work on Cross-lingual Toxicity Detection. Three different aspects of this problem were considered.

Cross-lingual combination: The first use of cross-lingual models is to offer the possibility of regrouping datasets from multiple languages to form a larger dataset with more training data, and potentially get better results on individual datasets. This is an extension of the work on dataset merging and multitask learning introduced in sections 4.2.2 and 4.2.3 (with results in table 4.8). These methods could work when providing additional data to a weak dataset, and a similar principle can be applied across languages, although combing datasets with too much content difference will still be detrimental. Results would be completely dependent on the choice of datasets and their combination, requiring an exhaustive search to determine the optimal combination. We could not find any cases were cross-lingual data combination was beneficial. This task was already challenging with monolingual models, but here the addition of a language gap makes this task even more difficult.

Zero-shot classification: The second aspect is zero-shot classification, with the goal of designing models able to classify data in a given language, while only possessing training data in other languages. This aspect is especially important for languages with small populations where no data are available, so that classification models can still be used by them.

Multilingual classification: Finally, as an extension of zero-shot classification, we might want to build a single multilingual model that can classify multiple languages simultaneously. It should work correctly on the language used for training, but also on unknown languages.

5.1 | Models

We experiment with two categories of models for the cross-lingual tasks: section 5.1.1 presents models that are purely multilingual, directly understanding data in multiple languages, while section 5.1.2 presents models that require data translation at some point in the pipeline. The difference between these two types of models is illustrated in figure 5.1.

5.1.1 | Purely multilingual models

We start with *purely* multilingual models that can utilise data from multiple languages without any additional preprocessing required. We use three types of purely multilingual models; models that use LASER [23] sentence embeddings, models that use MUSE [50] words embeddings, and models based on transformers (see section 4.1.4).

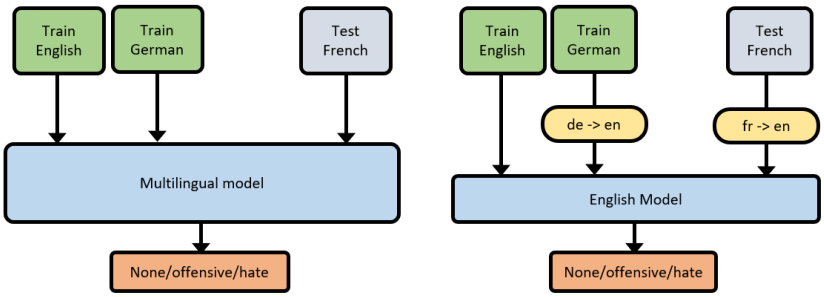

Figure 5.1. On the left, a purely multilingual architecture, on the right, a multilingual architecture that requires translations, illustrated here with English and German training data, and French test data. The purely multilingual model uses the data directly, while the other model needs to translate the German and French data into English to use the monolingual English model.

LASER embeddings

LASER embeddings [23] are multilingual sentence embeddings generated by a biLSTM encoder, trained on 93 languages as part of the encoder-decoder architecture illustrated in figure 5.2.

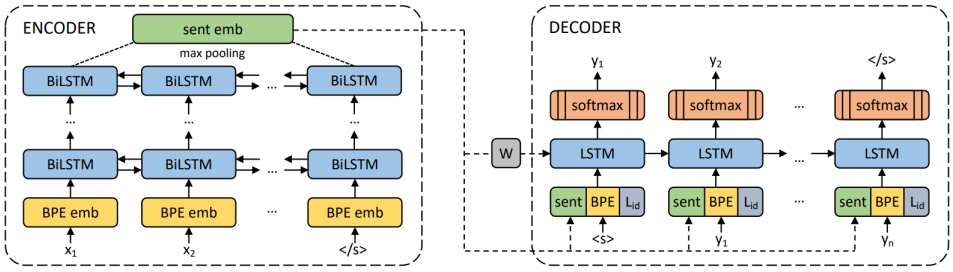

Figure 5.2. Architecture used to train the LASER embeddings encoder, illustration from Artetxe and Schwenk [23]

Byte-pair encoding [51] (see section 3.3.1) is used on the input data using a vocabulary acquired on the concatenation of all learning corpora (the encoder does not know the language of the input). A max-pool unit creates a sentence embedding using the output of a biLSTM network. The sentence embedding is then used in the decoder, trained to generate an English or Spanish translation of the original sentence, depending on the language specified in the input of the decoder. In our experiments, LASER embeddings are coupled with a multi-layer perceptron (MLP), a sequence of linear and non-linear layers. We use the *laserembeddings* Python library[1] to generate the embeddings. Unlike with the transformer models, there is no fine-tuning happening to the biLSTM network that generates the embeddings.

MUSE embeddings

MUSE embeddings [50] are multilingual word embeddings with support for 30 languages, generated by aligning FastText [11] word embeddings into a global embedding space using a bilingual dictionary that provides anchor points for the alignments of the two embeddings spaces. We use Gensim [34] to load them into the embedding layer from PyTorch [33], with embedding tuning disabled to avoid a dissociation of the embedding space of different languages in the case of zero-shot learning. Those embeddings are used in conjunction with the biLSTM model presented in section 4.1.3.

[1] https://github.com/yannvgn/laserembeddings

Multilingual transformer

For the transformer model we use XLM-RoBERTa [24], based on RoBERTa [15] and trained on 2.5TB of data from 100 languages. We connect it to the CNN classification head (section 4.2.1), as it usually performs well (see table 4.7), and using a common classification head will make the comparison between all models easier. The same CNN classification head is also used for all other transformer models in this chapter.

5.1.2 | Translation-based models

A second approach to cross-lingual classification is to translate all available data into a single language, and then use a monolingual model. In this section we present multiple architectures based on this idea.

Data translation

To generate the translations, we use MarianNMT [52], a neural machine translation model trained on the OPUS [53] parallel corpus by a research team at Helsinki University[2] [54]. We use a Python implementation of the model available in *huggingface's transformers* library.[3] All translations are done in advance on the cleaned data (cleaning according to section 4.3.2), and it takes approximately 0.1 to 0.2 seconds per sentence on a NVIDIA GeForce GTX 1080TI. We need a specific model for each language pair, but since there is not a model for every language pair[4] a series of translations is sometimes necessary. There are also a few models that understand language groups (e.g. Romance/Celtic/Nordic languages) that can be useful to avoid having to load too many translation models concurrently.

Before using the translations, it is important to know their quality. We show in table 5.1 a few experiments conducted with translations between English and French sentences. Censorship of extreme words was added for the report but was not present during the experiments.

Overall the translations are quite good and the original sense is usually maintained. For instance, the presence and position of the word *f*ck* have a large impact on the translated sentences (1-6). The structure of the sentence changes to accommodate this word, instead of simply having the literal translation inserted in the output. Some rare expressions and words are not translated by the model (7-9), and the English word is simply kept in French. In these cases, a French model that sees this word might still be able to learn some patterns, as this non-translation should happen in multiple cases. For some homonyms, the quality of the translation depends on the whole sentence, and the original meaning might only be preserved in specific cases, e.g. (10-11) where the presence of *"such"* has an impact on the final meaning. With other homonyms, the translation seems to work without issues (13-14). If we look at French sentences translated to English, we have similar patterns; adding an offensive word will usually fundamentally change the structure of the translation (17-20), it does work with some homonyms (21-26) and common expressions are correctly translated (27). Finally, there are a few cases where the translation just does not make sense (12, 16), but it usually happens with a single word, and adding more context helps (10, 15).

Overall these models seem to work quite well, at least with common sentences and insults, we therefore think that translation-based models have some potential.

[2] https://github.com/Helsinki-NLP/Opus-MT
[3] https://huggingface.co/transformers/model_doc/marian.html
[4] List of all language pairs listed on https://huggingface.co/Helsinki-NLP

#	Translation	Original	Translated
0	en → fr	I don't give a f*ck	J'en ai rien à fo*tre
1	en → fr	I hate you	Je te déteste
2	en → fr	I f*cking hate you	Je te déteste, p*tain
3	en → fr	Shut up	Taisez-vous
4	en → fr	Shut the f*ck up	Ferme ta gueule
5	en → fr	I broke my f*cking car	J'ai cassé ma p*tain de voiture
6	en → fr	I f*cking broke car	J'ai une p*tain de voiture cassée
7	en → fr	pikey	pikey
8	en → fr	You're a lazy sod	Tu es un sod paresseux
9	en → fr	He's a pillock	C'est un pilock
10	en → fr	She's such a h*e	C'est une sal*pard
11	en → fr	She's a h*e	C'est une houe
12	en → fr	h*e	C'est quoi, ça?
13	en → fr	Hang in there	Accrochez-vous
14	en → fr	Go hang yourself	Allez vous pendre
15	en → fr	He is an *sshat	C'est un enfoiré
16	en → fr	*sshat	- C'est pas vrai. - C'est pas vrai.
17	fr → en	Casse-toi de là c*nnard	Get the f*ck out of here
18	fr → en	Casse-toi	Get out of here
19	fr → en	Ca fait mal, p*tain	It hurts, damn it
20	fr → en	Ca fait p*tain de mal	It f*cking hurts
21	fr → en	C'est ch*ant	It's boring
22	fr → en	Il est ch*ant	He's a pain in the *ss
23	fr → en	Je t'emm*rde	F*ck you
24	fr → en	Il s'emm*rde	He's pissed off
25	fr → en	Il faut jeter les ordures	You've got to throw away the garbage
26	fr → en	T'es une ordure	you're a piece of sh*t
27	fr → en	Ta gueule	Shut up

Table 5.1. A few examples of sentences translated with the Opus-MT models.

Simple translation models

Using Opus-MT [54] we can now translate the available data into many languages, allowing for the use of the following architectures that utilise data translation and a monolingual transformer model with the CNN classification head (see section 4.2.1):

- translate all data into English and use RoBERTa [15] with the CNN classification head;
- translate all data into French and use FlauBERT [38] with the CNN classification head;
- translate all data into German and use german-BERT[5] with the CNN classification head.

We did not experiment with simpler models (Logistic Regression Classifier, CNN or biLSTM) in this translation approach, as we saw in the first part of this project that transformers models always outperformed them (comparison in table 4.12).

[5] https://deepset.ai/german-bert

5.2 | Improvements

Starting from the idea of using monolingual models with translations (section 5.1.2), we can create more advanced architectures where data translation is used to train multiple models in parallel. This idea is inspired by the work of Wan [21] and Pamungkas and Patti [22] and adapted to transformer models. We present here three different architectures that use translations and joint learning.

5.2.1 | Joint transformers

Table 4.6 showed that monolingual models usually outperform multilingual models, since they are trained on more data from the target languages, and they do not have to learn the specificities of multiple languages simultaneously. We propose here a joint architecture, *joint transformers*, where we use a monolingual transformer model (see section 4.1.4) for each language in the data. The goal is to utilise the advantages of monolingual models while still being compatible with multiple languages. Such a model is illustrated in figure 5.3.

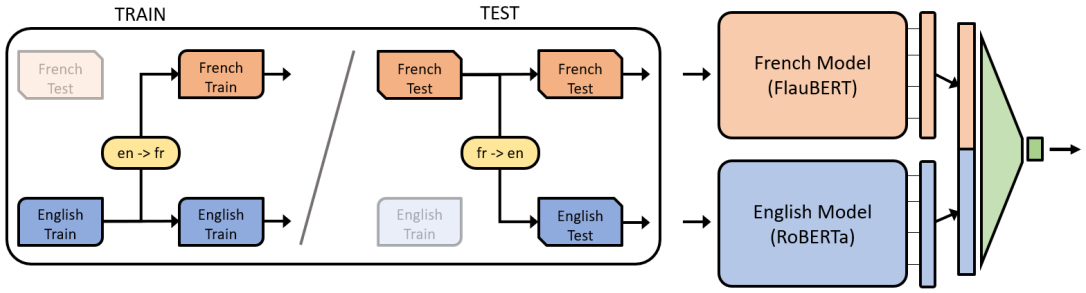

Figure 5.3. *joint transformer* architecture, here with English training data and French test data. We train an English transformer with the English data, but also a French transformer by translating English data into French. During the evaluation, the French model takes the French test data directly, and the English model takes the French data translated into English.

A monolingual transformer is initialised for each language of the data, and their outputs are sent into a single linear layer to make a prediction. During training, all data are translated into all other languages and the original and translated data are given to all transformers in parallel. All models receive a different representation of the original sentence and are trained jointly. For the evaluation the same process applies, with multidirectional data translations to feed all transformers in parallel.

The largest drawback of this architecture is its scalability, as we need a transformer model for every language in the data, but also need to perform all the multidirectional translations, and it becomes impractical for more than two or three languages. We can mitigate this problem by selecting two or three main languages that will have a transformer, and any additional language will be translated into the main languages. This technique is also useful if a language simply does not have a monolingual model.

5.2.2 | Joint LASER

The second proposed approach, *joint LASER*, is similar to the *joint transformers* model (section 5.2.1), except that we use a multi-layer perceptron with LASER sentence embeddings [23] for each language in the dataset in place of the transformers. Their outputs are also concatenated and sent to linear layers to make a prediction (see figure 5.4).

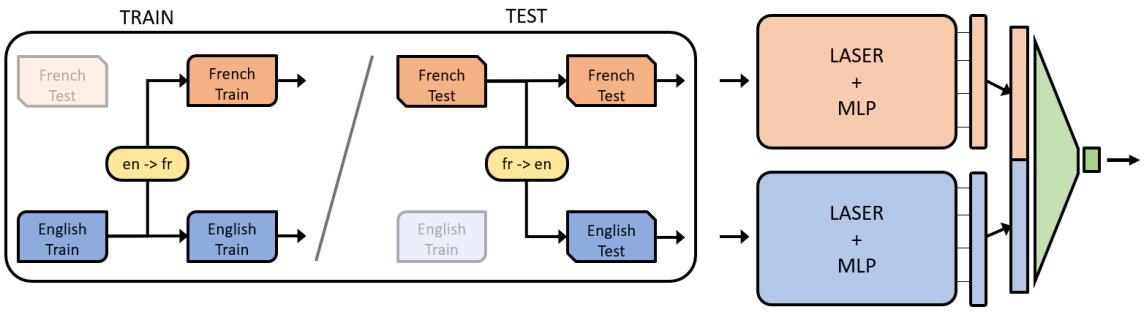

Figure 5.4. *joint LASER* architecture, with English train data and French test data. Each LASER+MLP module receives data from a unique language.

Both LASER+MLP blocks operate in a monolingual regime, but their outputs are still in the same embedding space. The scalability problem is less pronounced than with the *joint transformers*, due to the smaller size of LASER+MLP blocks. We also do not need to find monolingual models for each language, as we directly use multilingual models. If the target language is not supported by LASER embeddings (which should be rare as LASER embeddings support more than 93 languages), we can always translate the data into a supported language.

5.2.3 | Joint transformer and LASER

With the *joint transformers* and *joint LASER* architectures, similar models but in different languages are connected, hoping that the translations will generate multiple representations of the same sentence. We can also join completely different models together to achieve a similar effect. The base idea is similar to ensemble learning, in the sense that adding some variety to the models might result in better results, the difference being that the combination of all models is done more carefully than with a simple majority vote.

We propose to combine a transformer with a LASER+MLP block. Both will receive the same data and produce different representations of the input, then joined together to make a prediction. During training, the final layer, the transformer and the MLP are all updated together. For the transformer, there are two choices; use a monolingual transformer, in which case translations are required to send data to the transformer, or use a multilingual transformer model so that no translations are required. The LASER+MLP block will work in a multilingual regime by receiving all data directly. We illustrate these two variants in figures 5.5 and 5.6.

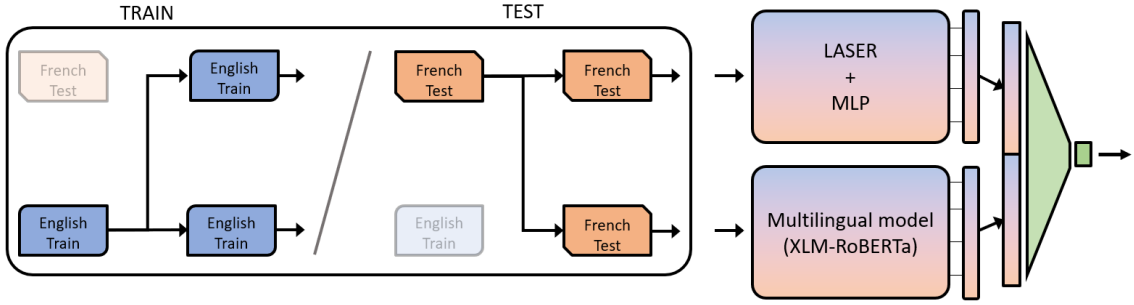

Figure 5.5. *joint transformer and LASER* architecture, variant with the multilingual transformer, with English train data and French test data. Both the LASER+MLP and transformer modules receive training and test data as-is, without any translations.

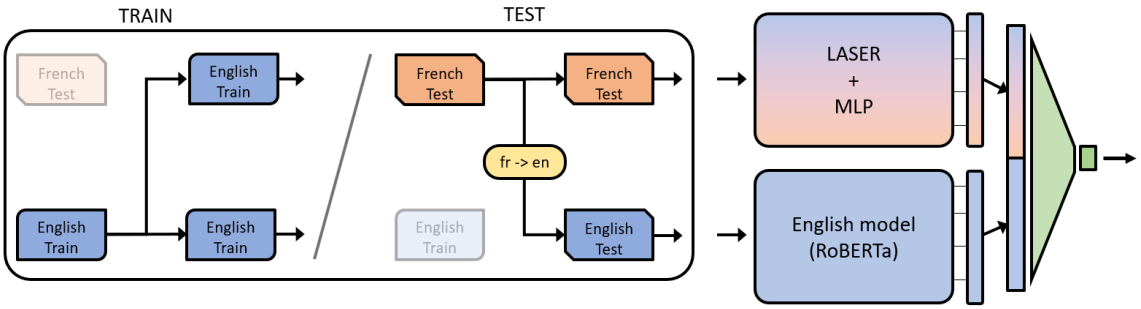

Figure 5.6. *joint transformer and LASER* architecture, variant with the monolingual transformer, with English train data and French test data. The monolingual RoBERTa transformer model only receives English data (original during training, translated from French during evaluation), and the LASER+MLP block will receive English data during training, and French data during evaluation.

Since we join two radically different models (transformer and LASER+MLP), each model should interpret the input with two different points of view, maybe focus on different concepts, and the concatenation of those representations should give a better summary for the sentence, and therefore a more accurate classification.

5.2.4 | Hurtlex

Iterating on the previous joint architectures, we propose variants to the simple translation models (section 5.1.2) and the *joint transformer and LASER* approach (section 5.2.3), where we also concatenate a representation of the presence of offensive and hateful words in the sentence.

We use Hurtlex[6] [55], a multilingual lexicon that classifies offensive and hateful words into 17 categories (cognitive disabilities and diversity, moral and behavioural defects, words related to social and economic disadvantage, animals, male genitalia, etc.) For each sentence, we create a vector of length 17 (the number of Hurtlex categories), indicating how many words from each category are in the sentence, and this vector is directly concatenated to the output of the transformer (and the output of the LASER+MLP block for the joint model). Since the Hurtlex categories are the same across all languages, this representation is multilingual and might be able to detect rare insults and swearwords whose sense might otherwise be lost in translation.

5.3 | Experiments

We now present results for the models introduced in sections 5.1 and 5.2 for the two different aspects of cross-lingual classification we are interested in: *zero-shot* classification and *multilingual* classification.

5.3.1 | Experimental setup

We report the averaged macro-F1 score on the test sets in our results, with the standard deviation over three runs, each with a different randomised *train*/*validation*/*test* split (with 80%/10%/10% proportions), and early stopping on the validation score. Unless specified otherwise, all other cleaning, training and architecture parameters are unchanged from the first part of the project (section 4.3).

[6] https://github.com/valeriobasile/hurtlex

5.3.2 | Zeroshot classification

In this section, we consider only one dataset for each language and try to improve the macro-F1 score for each language in a zero-shot approach. We are interested in obtaining the best possible score in a language by only using data from other languages for training. We only experiment with the Dachs datasets as we found that they were relatively clean, with good scores in French and English, and we expect the annotations and class distributions to be quite similar between the three languages as they come from the same authors. Excluding the other datasets from the start removes an important variable for the results (the selection of the best combination of datasets) and we can therefore concentrate on comparing the different architectures.

Multilingual models

We first experiment with the *purely* multilingual models, the ones that recognise input data in any language. We have a multi-layer perceptron (MLP) that uses LASER sentence embeddings [23], a biLSTM with MUSE word embeddings [50], and the multilingual XLM-RoBERTa transformer model [24] with the CNN classification head (all three models presented in section 5.1.1). There are also the joint approaches from sections 5.2.3 and 5.2.4, where the output of the transformer is joined to the output of the LASER+MLP block and/or the Hurtlex [55] representation. For each target language, we show in table 5.2 the macro-F1 scores obtained by training the model on either one or two of the other languages.

The XLM-R model uses the CNN classification head; we train it for 10 epochs with a learning rate of 2×10^{-5} for the transformer, a learning rate of 5×10^{-5} for the classification head, and the cross-entropy loss. The biLSTM has a hidden size of 100, and we train it for 50 epochs, with a learning rate of 10^{-4}, and the weighted cross-entropy loss). For the LASER+MLP model, we use a MLP with layers of sizes 128, 64 and 3, separated by LeakyReLUs with a slope of 0.01, a learning rate of 10^{-3} and the weighted cross-entropy loss. The embedding tuning for MUSE embeddings [50] is disabled, to avoid dissociation of the train and test embedding spaces. For the joint models, the classification head of the transformer has 64 outputs instead of the usual 3, and if applicable are concatenated to the LASER and/or Hurtlex representations (the LASER embeddings first go through a linear layer with 64 outputs), before going through a LeakyReLU with slope 0.1, and a final linear layer with 3 outputs. For the joint models, the learning rate of the transformer is 2×10^{-5} and the learning rate of all other layers is 5×10^{-5}. All models use the Adam optimiser [49].

Data		Model					
Test	Train	LASER+MLP	MUSE+biLSTM	XLM-R	XLM-R+L	XLM-R+HL	XLM-R+L+HL
D-EN	D-FR	55.4 (± 2.0)	45.7 (± 1.7)	59.5 (± 1.1)	60.0 (± 0.8)	**60.7** (± 1.0)	**60.6** (± 0.3)
D-EN	D-DE	55.4 (± 1.4)	43.8 (± 4.2)	43.3 (± 1.4)	**52.8** (± 1.2)	51.2 (± 2.2)	46.7 (± 2.1)
D-EN	D-FR+DE	53.3 (± 1.3)	49.0 (± 1.1)	57.6 (± 2.1)	**60.3** (± 0.5)	59.8 (± 1.3)	60.0 (± 0.6)
D-FR	D-EN	**59.5** (± 1.0)	39.8 (± 0.4)	53.3 (± 4.1)	56.1 (± 0.2)	52.0 (± 2.6)	52.6 (± 1.0)
D-FR	D-DE	**57.1** (± 1.9)	37.7 (± 2.5)	38.3 (± 1.6)	42.1 (± 1.2)	47.3 (± 2.9)	45.3 (± 1.7)
D-FR	D-EN+DE	**61.7** (± 1.1)	41.5 (± 1.6)	47.1 (± 2.5)	50.4 (± 2.5)	55.9 (± 0.5)	54.3 (± 2.3)
D-DE	D-EN	41.8 (± 2.4)	36.7 (± 0.8)	43.6 (± 2.4)	41.8 (± 1.8)	**45.2** (± 0.5)	**45.4** (± 0.7)
D-DE	D-FR	40.4 (± 1.4)	33.4 (± 1.5)	43.4 (± 1.5)	43.3 (± 1.3)	**45.0** (± 1.3)	**45.2** (± 1.1)
D-DE	D-EN+FR	38.4 (± 2.7)	35.5 (± 1.3)	45.4 (± 2.1)	44.8 (± 0.6)	45.1 (± 2.1)	**47.6** (± 0.1)

Table 5.2. Zero-shot results on Dachs, using the purely multilingual models. We first have the classical models in the first three columns, and the joint models in the remaining columns, where the XLM-R model is joined to a LASER+MLP block and/or the Hurtlex representation.

The MUSE embeddings do not work very well, with significantly lower scores than the other architectures, and for German test data it is as bad as a random prediction (see table 3.4). Despite having a simpler architecture, the LASER+MLP model is competitive with the transformer models, surpassing XLM-R in multiple cases. It seems that XLM-R encounters some difficulties when there are German data in the training sets; in both cases where we only train on German, the score on XLM-R is significantly lower than with the other languages for training. This large difference is not present with the other two models, nor in the opposite direction with German as the test language. Finally, when comparing the basic XLM-R model with the joint models, we see that the joint models are typically better. In all cases, there is at least one joint model performing better than the base architecture. The better model is not the same for all languages, but overall we see a clear benefit in using the joint architecture with additional representations.

Translation-based models

The second approach to zero-shot classification is to use translations and monolingual models. For the translations into English, French and German, we use RoBERTa [15], FlauBERT [38] and german-BERT[7], respectively (they are the best *monolingual* models in table 4.6, we did not use XLM-R for German to avoid mixing conceptually different models). We also test the *joint transformers* architecture (section 5.2.1). For this model, in the experiments with three languages, we could not use the original transformer models due to memory limitations. We used distilRoBERTa, FlauBERT-small and distilBERT-german instead of the original models,[8] which puts these models at a small disadvantage. We compare all translation-based models with the best purely multilingual models taken from table 5.2, and show all results in table 5.3.

Data		Model				
Test	Train	Best Multi-L	EN+RoBERTa	FR+FlauBERT	DE+gBERT	Joint
D-EN	D-FR	59.5 (± 1.1)	**63.1** (± 2.0)	59.1 (± 1.6)	55.6 (± 0.8)	62.9 (± 0.6)
D-EN	D-DE	**55.4** (± 1.4)	54.8 (± 1.7)	43.1 (± 2.8)	38.3 (± 0.6)	52.5 (± 2.6)
D-EN	D-FR+DE	57.6 (± 2.1)	**62.8** (± 0.8)	58.3 (± 0.8)	55.3 (± 1.3)	62.1 (± 1.0)
D-FR	D-EN	59.5 (± 1.0)	57.6 (± 1.3)	**62.5** (± 2.5)	54.3 (± 0.5)	58.6 (± 2.7)
D-FR	D-DE	**57.1** (± 1.9)	50.0 (± 3.6)	56.2 (± 3.5)	33.8 (± 1.6)	42.0 (± 1.1)
D-FR	D-EN+DE	**61.7** (± 1.1)	57.7 (± 1.2)	60.5 (± 4.1)	53.8 (± 1.4)	57.7 (± 2.4)
D-DE	D-EN	43.6 (± 2.4)	45.3 (± 2.1)	44.8 (± 2.8)	42.2 (± 0.7)	**45.7** (± 0.3)
D-DE	D-FR	43.4 (± 1.5)	**45.7** (± 1.4)	*44.2* (± 0.8)	42.1 (± 2.2)	43.2 (± 1.1)
D-DE	D-EN+FR	45.4 (± 2.1)	46.4 (± 0.8)	45.9 (± 0.8)	44.5 (± 1.1)	**47.6** (± 1.8)

Table 5.3. Results for translation-based models on Dachs. *Best Multi-L* is the best purely multilingual model, we then have the three simple translation models in English, French and German, and *Joint*, the *joint transformers* model with one transformer for each language.

Despite some potential noise and loss of meaning due to the translations, the models that use translations perform better than purely multilingual models in multiple cases. There are some exceptions when there are German training data, but in those cases there is usually a model using translation not too far behind. Interestingly, the language chosen for the translation has some importance; when we want to classify English data, the best choice is to train an English model with English translations, and when we want to classify French data, the best choice is

[7] https://deepset.ai/german-bert
[8] More details on these models on https://huggingface.co/transformers/pretrained_models.html

to use a French model and French translations. This pattern does not apply to the experiments with German test data, probably because german-BERT is a simpler model than RoBERTa and FlauBERT, with less training data used in the original pretraining (for the same reason that XLM-R was better than german-BERT on German data in table 4.6). The results with the *joint transformers* architecture are not conclusive, with no significant gain in performance especially when considering the increased cost in resources, but it might be caused by the simplified transformers used due to technical limitations.

Joint LASER: We report in table 5.4 results for the second joint learning approach, *joint LASER* (section 5.2.2), with one LASER+MLP block used for each language in the data and multidirectional translations. We compare those with the simple multilingual LASER+MLP model from table 5.2. Here, the LASER embeddings from each language each go trough a linear layer with 128 outputs, a LeakyReLU with slope 0.01, and a dropout layer with probability 0.1, before being concatenated and going through a linear layer with 64 outputs, a LeakyReLU with slope 0.01 and a final linear layer with 3 outputs, one for each class. We use a learning rate of 10^{-3} and the weighted cross-entropy loss.

Data		Model	
Test	Train	LASER+MLP	Joint LASER+MLP
D-EN	D-FR	55.4 (± 2.0)	**57.5** (± 1.1)
D-EN	D-DE	55.4 (± 1.4)	**56.4** (± 0.3)
D-EN	D-FR+DE	53.3 (± 1.3)	**57.2** (± 0.7)
D-FR	D-EN	59.5 (± 1.0)	**61.1** (± 1.2)
D-FR	D-DE	57.1 (± 1.9)	**57.9** (± 0.8)
D-FR	D-EN+DE	61.7 (± 1.1)	**63.0** (± 0.8)
D-DE	D-EN	41.8 (± 2.4)	**45.6** (± 1.3)
D-DE	D-FR	40.4 (± 1.4)	**42.3** (± 0.9)
D-DE	D-EN+FR	38.4 (± 2.7)	**44.8** (± 0.7)

Table 5.4. *Joint LASER* models compared to the individual LASER+MLP multilingual models, on the Dachs datasets

For all cases, we have a non-negligible improvement over the original LASER model, while keeping a relatively simple model (still less complex than the transformer models). We add the requirement for data translation, and the complexity of the model doubles, but we think it is a good trade-off when looking at the improvements we have in some cases. The model only needs to learn the parameters for a few linear layers, which can be done rapidly on a GPU, although we still need to generate the LASER embeddings, which can take some time. For most cases, the transformer models in table 5.3 still perform better, but we are able to get quite close in multiple scenarios, with significantly lower training time.

Joint transformer + LASER + Hurtlex: Finally, table 5.5 shows results for the third type of joint models, where we join the outputs of a transformer model with a LASER+MLP block and/or the Hurtlex [55] representation (sections 5.2.3 and 5.2.4). For both RoBERTa and FlauBERT, with English or French translations, we show results for the transformer by itself, the transformer with LASER embeddings, the transformer with the Hurtlex representation and the transformer with both LASER embeddings and the Hurtlex representation. We do not perform this experiment with german-BERT,[9] as we saw in table 5.3 that it is always worse than the English and French models.

[9] https://deepset.ai/german-bert

Data		EN + RoBERTa				FR + FlauBERT			
Test	Train	T	T+L	T+HL	T+L+HL	T	T+L	T+HL	T+L+HL
D-EN	D-FR	63.1 (± 2.0)	63.7 (± 0.6)	**64.7** (± 0.3)	63.0 (± 1.9)	59.1 (± 1.6)	59.0 (± 1.3)	60.1 (± 0.9)	59.0 (± 0.6)
D-EN	D-DE	54.8 (± 1.7)	56.2 (± 1.4)	56.5 (± 1.5)	**57.5** (± 1.0)	46.5 (± 6.1)	46.2 (± 3.2)	47.9 (± 4.0)	44.9 (± 3.0)
D-EN	D-FR+DE	62.8 (± 0.8)	**64.5** (± 0.8)	63.9 (± 1.3)	63.7 (± 0.1)	58.3 (± 0.8)	58.0 (± 1.3)	57.1 (± 3.1)	57.9 (± 0.5)
D-FR	D-EN	57.6 (± 1.3)	59.3 (± 3.6)	56.8 (± 1.1)	58.6 (± 3.3)	62.5 (± 2.5)	**66.3** (± 1.5)	65.6 (± 0.5)	63.7 (± 2.9)
D-FR	D-DE	50.0 (± 3.6)	51.1 (± 2.9)	51.5 (± 4.0)	52.4 (± 4.0)	**56.2** (± 3.5)	**56.3** (± 5.4)	**57.6** (± 4.0)	54.7 (± 0.8)
D-FR	D-EN+DE	57.7 (± 1.2)	55.6 (± 2.6)	56.3 (± 1.7)	56.8 (± 0.8)	60.5 (± 4.1)	66.0 (± 0.5)	62.3 (± 3.9)	**66.9** (± 2.1)
D-DE	D-EN	45.3 (± 2.1)	44.6 (± 2.1)	**48.3** (± 1.5)	45.5 (± 3.5)	44.8 (± 2.8)	46.1 (± 0.5)	45.2 (± 1.8)	45.8 (± 2.0)
D-DE	D-FR	45.7 (± 1.4)	45.4 (± 1.6)	**49.5** (± 2.6)	46.3 (± 2.2)	44.2 (± 0.8)	46.5 (± 1.3)	43.7 (± 0.8)	43.3 (± 0.8)
D-DE	D-EN+FR	46.4 (± 0.8)	49.2 (± 2.0)	**50.2** (± 1.3)	49.4 (± 0.6)	45.9 (± 0.8)	46.2 (± 3.0)	47.2 (± 0.8)	48.0 (± 2.1)

Table 5.5. Comparison between the transformer models with translation (*T*), and the models where the transformer is joined with the LASER+MLP block (*T+L*). We also show the variants with the Hurtlex representation concatenated (*T+HL* and *T+L+HL*).

Using additional representations jointly with the transformer let us improve all results over the base models, but the best variant depends on the languages used. On average, the Hurtlex representation seems to be slightly more beneficial to the score than the LASER embeddings, but as usual there is a lot of variance with some models, making confident conclusions difficult. Joining both the Hurtlex and LASER representations is usually not useful.

We summarise all *zero-shot* results in table 5.6, reporting the best results for purely multilingual models and for models with translations. We consider those two approaches different enough to warrant separate reporting. Translation-based models require more processing before training the model, and most importantly require constant translation of new data while using them in practice. To support multiple languages, multiple Opus-MT [54] translation models will be required per language, which can get costly very quickly. We do not have this problem with the purely multilingual model.

Data		Mulitlingual		Translation	
Test	Train	Model	Score	Model	Score
D-EN	D-FR	XLM-R+HL	60.7 (± 1.0)	RoBERTa+HL	**64.7** (± 0.3)
D-EN	D-DE	XLM-R+L	52.8 (± 1.2)	RoBERTa+L+HL	**57.5** (± 1.0)
D-EN	D-FR+DE	XLM-R+L	60.3 (± 0.5)	RoBERTa+L	**64.5** (± 0.8)
D-FR	D-EN	LASER+MLP	59.5 (± 1.0)	FlauBERT+L	**66.3** (± 1.5)
D-FR	D-DE	LASER+MLP	57.1 (± 1.9)	Joint LASER	**57.9** (± 0.8)
D-FR	D-EN+DE	LASER+MLP	61.7 (± 1.1)	FlauBERT+L+HL	**66.9** (± 2.1)
D-DE	D-EN	XLMR-R+L+HL	45.4 (± 0.7)	RoBERTa+HL	**48.3** (± 1.5)
D-DE	D-FR	XLMR-R+L+HL	45.2 (± 1.1)	RoBERTa+HL	**49.5** (± 2.6)
D-DE	D-EN+FR	XLMR-R+L+HL	47.6 (± 0.1)	RoBERTa+HL	**50.2** (± 1.3)

Table 5.6. Best results on zero-shot classification using multilingual models only, and using translations with multilingual and monolingual models.

In all cases, the models that utilise translations perform better than the multilingual models. The best models are also usually joint models; the use of diverse representations of the input is clearly useful. It is however important to remember that this is an ideal case, where the data used for zero-shot evaluation is sometimes in the same language as the monolingual transformer, and that translations to English and French are probably better than average due to the common use of these languages.

We conclude this section with detailed results for the best models from table 5.6. We show those results in table 5.7, alongside the detailed results of the monolingual models on each language (using models from table 4.7). There is still a large gap between the scores obtained on the monolingual classification, and what can be achieved with zero-shot classification. The recall on `hate` and `offensive` is particularly low, with the largest difference in French. On Dachs DE, however, we get closer results; since the dataset itself is not very good, it is less problematic to use other data for training. With French and English, the domain gap has more impact.

Data				None		Offensive		Hate	
Test	Train	Model	m-F1	P	R	P	R	P	R
D-EN	D-FR	RoBERTa+HL	64.7	82.7	82.7	57.4	58.7	57.3	50.7
D-EN	D-DE	RoBERTa+L+HL	57.5	77.2	90.7	63.3	39.5	55.5	32.4
D-EN	D-FR+DE	RoBERTa+L	64.5	81.7	83.3	58.6	54.2	53.4	56.5
D-EN	*D-EN*	*RoBERTa*	*74.9*	*86.6*	*88.7*	*70.5*	*67.3*	*72.6*	*64.8*
D-FR	D-EN	FlauBERT+L	66.3	85.4	90.8	62.9	53.8	62.2	47.0
D-FR	D-DE	Joint LASER+MLP	57.9	83.3	82.0	49.1	48.5	40.1	47.6
D-FR	D-EN+DE	FlauBERT+L+HL	66.9	84.4	93.4	67.5	49.3	67.5	47.2
D-FR	*D-FR*	*FlauBERT*	*78.7*	*91.0*	*89.9*	*68.7*	*71.9*	*77.4*	*73.7*
D-DE	D-EN	RoBERTa+HL	48.3	91.3	95.9	38.2	25.3	46.7	14.1
D-DE	D-FR	RoBERTa+HL	49.5	92.4	90.5	29.1	39.0	31.7	18.8
D-DE	D-EN+FR	RoBERTa+HL	50.2	92.3	93.0	34.7	35.4	36.9	17.9
D-DE	*D-DE*	*XLM-R*	*52.7*	*92.7*	*95.1*	*44.2*	*35.2*	*37.7*	*22.6*

Table 5.7. Detailed results for the best models on the zero-shot classification task on the Dachs datasets. The results are compared to monolingual models where we train and evaluate on the same dataset (in italic).

5.3.3 | Multilingual classification

In this section, the goal is to develop a unique model compatible with multiple languages that delivers comparable performance to individual monolingual models, while also performing well on other unknown languages. We train multiple models on English, French, and German data and evaluate them on these three languages but also additional languages with no training data.

For the training languages, we keep half of the datasets for each language that are in our opinion the best. For English we keep and merge Jigsaw and Dachs EN, as both the Davidson and Founta datasets have problems with the distinction between `hate` and `offensive`. We keep Dachs FR for French and Germeval for German. We create a balanced validation set, consisting of one third English data, one third French data and one third German Data, that we use for early stopping, but the evaluation on the test sets is still done for each language independently.

We use the same models as in zero-shot classification (section 5.3.2): the multilingual XLM-R and LASER+MLP models (section 5.1.1), the monolingual models using translations (Translate EN + RoBERTa, Translate FR + FlauBERT, section 5.1.2) and the joint variants with the LASER and/or Hurtlex representations joined to the transformers (section 5.2.3). We also show the *joint transformers* model (section 5.2.1) and *joint LASER* model (section 5.2.2), both with three modules (one for each training language). In table 5.8 we show the macro-F1 score on the three training languages, but also on additional datasets in unseen languages (Dachs GR, Dachs ES and Indonesian, see section 3.1).

Train on EN+DE+FR	EN	DE	FR	avg	ID	D-ES	D-GR	avg
LASER+MLP	69.0 (± 1.5)	53.1 (± 1.9)	57.6 (± 3.0)	59.9	**40.3** (± 0.9)	35.0 (± 0.9)	47.9 (± 0.3)	41.1
XLM-R	75.8 (± 2.6)	**61.4** (± 2.9)	76.4 (± 1.0)	71.2	26.1 (± 1.3)	44.5 (± 0.5)	53.0 (± 0.7)	41.2
XLM-R+LASER	76.8 (± 2.1)	**61.9** (± 2.1)	74.7 (± 1.4)	71.1	30.6 (± 0.8)	41.6 (± 1.3)	53.6 (± 0.9)	41.9
XLM-R+HL	74.7 (± 0.7)	60.5 (± 3.2)	76.5 (± 1.0)	70.6	28.2 (± 0.2)	44.1 (± 2.2)	52.8 (± 1.4)	41.7
XLM-R+HL+LASER	75.8 (± 0.7)	60.6 (± 2.7)	**78.4** (± 1.1)	**71.6**	27.0 (± 0.5)	42.1 (± 0.7)	51.5 (± 0.5)	40.2
EN	77.8 (± 1.8)	57.5 (± 3.3)	69.2 (± 1.2)	68.2	32.7 (± 1.4)	44.8 (± 0.6)	52.4 (± 1.2)	43.3
EN+LASER	78.0 (± 0.6)	57.9 (± 1.8)	69.5 (± 2.3)	68.5	34.8 (± 1.1)	43.1 (± 1.2)	53.6 (± 1.0)	43.8
EN+HL	77.8 (± 1.4)	57.2 (± 1.7)	71.1 (± 1.7)	68.7	32.6 (± 0.9)	44.3 (± 0.9)	53.3 (± 0.7)	43.4
EN+HL+LASER	**78.9** (± 0.8)	56.9 (± 3.5)	70.8 (± 1.5)	68.9	31.1 (± 0.6)	44.1 (± 0.9)	55.3 (± 0.7)	43.5
FR	72.4 (± 2.9)	54.6 (± 1.7)	78.2 (± 0.3)	68.4	32.4 (± 1.4)	40.6 (± 1.4)	53.6 (± 2.7)	42.2
FR+LASER	74.4 (± 1.3)	54.7 (± 2.0)	75.5 (± 3.0)	68.2	33.5 (± 1.1)	42.5 (± 1.3)	53.7 (± 1.4)	42.2
FR+HL	73.4 (± 1.8)	54.8 (± 1.3)	76.1 (± 1.9)	68.1	33.0 (± 1.1)	40.1 (± 1.1)	**55.7** (± 1.5)	42.9
FR+HL+LASER	73.0 (± 0.9)	54.7 (± 0.4)	**78.8** (± 0.9)	68.8	32.2 (± 0.2)	41.5 (± 0.3)	53.9 (± 0.5)	42.5
joint LASER	71.0 (± 1.3)	53.8 (± 0.4)	60.4 (± 1.9)	61.7	36.7 (± 0.2)	34.9 (± 3.2)	52.1 (± 0.5)	41.2
joint transformers	75.1 (± 3.5)	60.0 (± 2.2)	72.0 (± 0.2)	69.0	27.1 (± 1.1)	**50.8** (± 2.5)	54.8 (± 0.7)	**44.2**
Best individual models	78.1 (± 0.2)	61.6 (± 2.3)	78.3 (± 0.5)					

Table 5.8. Comparison of multilingual models trained on English, German and French data merged. We report the scores on the training languages, with the average, and the scores on Indonesian, Spanish and Greek data, with the average. The first group shows purely multilingual models, LASER+MLP and XLM-R with its joint variants. The second group shows the translation-based models: translate to English and use RoBERTa or translate to French and use FlauBERT, with all the joint variants, and finally the *joint LASER* and *joint transformers* models. For the training languages we show the scores when training an independent model for each language (using RoBERTa, FlauBERT or XLM-R for English, French and German data).

For the results on the training languages, we observe similar behaviours as in zero-shot classification (section 5.3.2); in English and French the translation-based methods are overall better, and the chosen translation language is important. In German, the XLM-R model is this time better than any of the translation models. For both categories of models, the joint variants are generally better than the base models, although what variant is the best depends on the language. On average over the training languages, the XLM-R models are better than the translation-based methods (mostly due to the German scores), with a slight advantage for the model joined with the LASER and Hurtlex representations. The LASER+MLP models are clearly worse than the transformer models, but the advantage of the *joint LASER* model is still present. Finally, the scores on some of the joint models are equivalent to the best individual monolingual models, showing the usefulness of the joint methods that could also be applied to a monolingual context. However, there is not one single multilingual model reaching maximal performance on the three languages, so for optimal results on these languages one would still need to use multiple individual models.

For the additional languages evaluated in a zero-shot context, it is quite hard to formulate a definitive conclusion of which model is the best. On Indonesian for instance, the single LASER+MLP model is the only one that reaches results better than a random prediction. For Spanish, the *joint transformers* is by far the best model, with not a lot of difference between the purely multilingual and translations-based models. In Greek the translation models are usually better. For the three languages, the additional LASER and/or Hurtlex representation are often useful, but overall the scores are very low, especially compared to English and French, making these models not really usable in practice.

Binary classification

Since some of the misclassifications come from a confusion between `hate` and `offensive`, we look at the binary classification scores (`toxic` or `non-toxic`) to get a better grasp of the performance of the models. We use the exact same models, but regroup the `hate` and `offensive` predictions into a unique `toxic` class, which now also let us evaluate three new Jigsaw datasets (Spanish, Italian and Turkish, presented in section 3.1). We show the macro-F1 scores in table 5.9.

Train on EN+DE+FR	ID	D-ES	D-GR	J-IT	J-ES	J-TR	avg
LASER+MLP	**62.9** (± 0.7)	45.6 (± 0.7)	69.7 (± 0.7)	69.3 (± 2.4)	69.3 (± 1.8)	76.2 (± 4.2)	65.5
XLM-R	47.0 (± 1.6)	52.7 (± 0.4)	74.1 (± 0.3)	67.8 (± 1.6)	70.8 (± 3.2)	73.4 (± 1.1)	64.3
XLM-R+LASER	55.5 (± 0.6)	50.8 (± 0.7)	74.7 (± 0.5)	73.2 (± 2.0)	72.1 (± 4.3)	79.7 (± 2.8)	**67.7**
XLM-R-HL	50.0 (± 1.9)	52.5 (± 1.1)	**73.7** (± 0.4)	68.8 (± 1.4)	69.3 (± 4.7)	78.1 (± 4.0)	65.4
XLM-R-HL-LASER	48.2 (± 1.5)	52.1 (± 0.6)	72.9 (± 0.6)	70.2 (± 6.9)	66.4 (± 1.0)	72.5 (± 2.7)	63.7
EN	53.3 (± 0.5)	56.5 (± 1.4)	71.4 (± 0.7)	68.8 (± 0.6)	67.8 (± 2.9)	72.6 (± 2.4)	65.1
EN-LASER	56.8 (± 0.6)	55.3 (± 1.6)	73.5 (± 0.1)	70.9 (± 1.1)	71.6 (± 2.5)	79.1 (± 4.8)	**67.9**
EN-HL	55.6 (± 1.1)	54.6 (± 0.8)	72.7 (± 0.2)	68.4 (± 2.4)	68.0 (± 3.4)	72.3 (± 2.0)	65.3
EN-HL-LASER	53.1 (± 0.2)	55.2 (± 0.4)	72.7 (± 0.4)	71.5 (± 2.9)	68.3 (± 2.2)	73.2 (± 1.6)	65.7
FR	54.9 (± 0.3)	50.8 (± 1.1)	71.9 (± 1.3)	69.5 (± 2.3)	70.6 (± 3.9)	77.2 (± 2.1)	65.8
FR-LASER	54.4 (± 0.5)	53.4 (± 0.5)	73.2 (± 0.9)	67.6 (± 1.5)	**73.7** (± 4.5)	**80.4** (± 5.4)	67.1
FR-HL	54.3 (± 1.2)	50.6 (± 0.4)	71.5 (± 0.8)	69.5 (± 1.4)	69.8 (± 2.3)	79.5 (± 1.4)	65.9
FR-HL-LASER	54.1 (± 0.4)	52.9 (± 0.1)	72.8 (± 0.6)	**71.7** (± 4.0)	71.4 (± 3.0)	77.0 (± 3.9)	66.6
joint LASER	58.6 (± 1.1)	46.7 (± 3.1)	71.3 (± 0.5)	68.5 (± 1.5)	67.3 (± 0.2)	79.6 (± 2.8)	65.3
joint transformers	46.9 (± 2.7)	**59.8** (± 2.8)	72.0 (± 0.6)	69.9 (± 1.9)	66.8 (± 1.7)	78.6 (± 1.6)	65.7

Table 5.9. Scores for the binary classification of toxic content on languages with no training data, with the averages over all languages. We use the the models from table 5.8.

We observe similar patterns to those of table 5.8; LASER+MLP is usually the worst model, and for multiple languages we can improve the score of the transformer models by joining additional representations. Altogether, the best approach seems to be to translate all data into English, and then use a RoBERTa model jointly with LASER embeddings. This gives us the second-best average in table 5.8 and best average in table 5.9. For a purely multilingual model, XLM-R joined with LASER embeddings is the best option and is actually very close to the performance of the translation models on the binary classification task, and most crucially does not require any translations. The final choice will depend on the languages that need to be supported, the exact classification task (two or three classes), and the available resources.

This time the Indonesian data score better than the Spanish data from Dachs with most models, a good indication that many misclassifications came from a confusion between `hate` and `offensive`. Perhaps most importantly, the impact of the choice of datasets is very clear; Dachs ES and Jigsaw ES, while both in the same language, have vastly different scores. It is hard to know exactly the cause of this difference; it can be a problem with the quality of annotations in Dachs ES or a larger domain gap from Dachs ES to the training data than from Jigsaw ES. This shows that it is extremely challenging to predict the performance of our models on general data, as all of the scores are more dependant on the dataset chosen rather than the language or model selected. Nevertheless, since we evaluated a lot of datasets, some patterns and conclusions should still apply relatively well to other datasets. With the three datasets from Jigsaw we also observe higher variations in the scores; this happens because we have less data for them, therefore taking a random test set from them will result in more variance, but we still think that the differences between the values are large enough to draw some conclusions.

6 | Discussion

In this chapter, we study in greater details a few aspects of the toxicity detection task to highlight some weaknesses of the models presented in the previous two chapters. We also present a few additional relevant experiments.

6.1 | Error Analysis

We consider here a few incorrectly classified messages to highlight factors that lead to erroneous predictions. There are two primary causes: there can be problems with the datasets and problems with the models. We mostly look at errors caused by the datasets, as they are easier to uncover and understand.

The major problem with some of the available datasets is the inconsistency between the annotations of various sentences. Two very similar sentences can sometimes be found in multiple classes. As a first example, we have the following sentences:

> *geta life f*ggot* → `hate`
> *stick to hockey f*ggot. you're finished* → `offensive`

The term *f*ggot* does not seem to be used as an attack against homosexual people here and should be in the `offensive` class in both cases according to the definitions presented in section 1.2. It can be argued about whether or not the use of this term should be considered `hate` or `offensive` in this context, but it should be the same class for both sentences here since the term is used in the exact same way. Similarly, we consider the following sentences:

> *what a f*cking ret*rd* → `offensive`
> *lmao madonnas music sucks? yeah you're f*cking stupid if you think gaga is better. ret*rd* → `hate`

In both cases the term *ret*rd* is used as a general insult and does not seem to target people with mental disabilities, but they were assigned to different labels. In more extreme cases, sentences that were extremely similar also had inconsistent annotations:

> *@latimes Go back to your own country* → `hate`
> *@FoxNews Nope. Go back to your own country* → `none`
> *@Newsweek Go back to your own country. #BuildTheWallNow* → `none`

All three messages come from the Dachs dataset, where each message only has one annotator, meaning that personal opinions and bias will have a huge effect in the annotations. The models will then be given contradictory information, hurting their performance.

We were also able to find some annotations that in our opinion were simply incorrect, for instance:

> *I hate aubergines* → `hate`
> *Just need to make it through the next weeks and then I can relax and by relax I mean start parking to move...* → `hate`

Manual research of these tweets on Twitter showed no special context that might indicate hate speech. Both messages were classified as `none` by our model.

All of those errors and inconsistencies in the annotations makes it hard to precisely know the performance of our models, and if misclassifications are caused by problems with the annotations or the model. The amount of such problems depends on the dataset, as the annotation methodologies and number of annotators are not always the same (see section 3.1), but it is impossible to know the full extent of this problem without a time-consuming review of all annotations.

The second type of misclassification happens when there is simply not enough training data to learn the toxicity of some content. For instance,

> *burn in h3ll s8n*

was classified as `none` by our model, simply because it never observed the words *h3ll* and *s8n* written this way before, therefore cannot infer anything from them. Similarly, when we have sentences with words that are rarely present in the training set, it is challenging to infer the correct class. For instance, the word *slits* in the sentence

> *i'm surprised you could read it with slits for eyes*

only appear three times in the Davidson dataset, above example included, and it is associated twice with the `none` class. More data are required to make the co-occurrence of *slits* and *eyes* recognisable as racism against Asian people.

Even with enough data to learn all currently hateful and offensive expressions, there would still be problems with unknown expressions. New words and expressions are sometimes created, especially on the internet, and with them novel ways to be toxic. Moreover, the meanings of some terms change over time and some become offensive, such as the word *Boomer* now considered as ageist and offensive by some people due to the rise in popularity of the expression "*OK Boomer*" used to mock some behaviours of people from the Baby Boomer generation. The same is true with *Karen*, nothing more than a name a few years ago but now a mocking term for a certain type of women perceived as entitled. These words, almost not present in our datasets, are now very common in online communities, therefore the models trained here will probably not properly detect this type of offensive content. The opposite can also happen, with terms like *queer* or *geek* that used to be pejorative now accepted by the communities they were once attacking[1].

Finally, it is extremely hard to define offensive and hateful words. If a word is considered offensive by only 1% of the population, should it be universally considered offensive? There is ordinarily no official authority that defines offensive words, so any classifier system that relies on manually annotated data will depend on the personal opinion of the annotators and authors, and the context in which those messages were written, thus it is extremely important for the data used to represent the general opinion of the population the model will be applied to.

[1] Examples of linguistic reappropriation: https://en.wikipedia.org/wiki/Reappropriation

6.2 | Domain gap

The first problem that contributes to relatively low scores when performing zero-shot classification remains the insufficient quality of some of the datasets; if we do not have a good score on a dataset when training it directly, we will not be capable of creating a decent zero-shot model with it. Not a lot can be done to fix this, except giving particular attention to dataset selection.

In the case of cross-lingual classification, there is on top of that the problem of the language gap. Some expressions might be considered rude in one language, but when translated literally will lose the original sense and any negative connotation. In practice, the quality of Opus-MT translations [54] seems reasonably good (as shown in table 5.1) but there are always exceptions that might lower the performance of the models.

Most importantly, as shown in section 3.1.2, the themes present in the different datasets are fairly different, which causes a non-negligible domain gap. In some datasets we have a lot of racism, in others it is more general hate (e.g. wish for harm). As an experiment, we tried zero-shot classification on English data only using a simple RoBERTa [15] with the default CLS classification head (see section 4.2.1). We show results in table 6.1.

Train	Test	macro-F1
Jigsaw	Jgsaw	83.7 (± 0.4)
Dachs EN	Dachs EN	74.9 (± 0.1)
Davidson	Davidson	77.8 (± 1.3)
Jigsaw	Dachs EN	56.7 (± 1.7)
Dachs EN	Jigsaw	55.3 (± 0.9)
Davidson	Dachs EN	43.4 (± 0.6)
Dachs EN	Davidson	41.7 (± 1.3)

Table 6.1. Zero-shot classification on some English datasets, using a simple RoBERTa model. The first three rows show each dataset's individual score, the other rows show zero-shot results.

It is clear that the domain gap between the datasets exerts a considerable influence on the scores. Each dataset has an individual score that is relatively high, a sign that the dataset itself is quite good and has sufficient data. However, with zero-shot classification the results are clearly lower, and it is extremely hard to utilise knowledge acquired in one dataset to classify other data. This is also the main cause behind the lack of good results with dataset merging (section 4.2.2).

Even with similar types and distributions of `hate` and `offensive` content in all datasets, there is still a difference in subjects and targets across languages. In the English (American) datasets, racism might typically be directed towards the Mexican and African-American populations, while in the French and German datasets racism might instead focus on North-African and Middle-Eastern countries. It will be challenging for a model trained uniquely on French data to learn the concept of hate against Mexicans. There are also subjects that despite being present in multiple datasets/regions do not have the same perceived offensiveness; for instance attacks against homosexuals will be looked at completely differently in Sweden or Iran. If we then mix the two sources, a model will have to employ contradictory data. This gap is present in all societal domains, from politics to religion and economy, which makes it extremely challenging to successfully combine datasets gathered in different countries.

All these examples fall into the domain gap category, and we think that this is the most critical problem and limitation to any model that regroups different datasets (in particular zero-shot classification models). The language gap is mostly crossed by the multilingual models or the translations, but there is not a lot that can be done about the domain gap. If we were to test these models on datasets in multiple languages, but from a similar geographical region and with the same collection methodology, we think that the scores might be better; we could for instance gather French, German and Italian comments on the respective *20 Minutes* (a Swiss newspaper) comment sections, in which case the themes of each dataset should be quite close.

To illustrate the impacts of the language gap and domain gap, we performed a few experiments. Supposing a dataset A in language L_A and a dataset B in language L_B, we train a XLM-R+LASER multilingual model (section 5.6) in four different ways, each of them corresponding to a different combination of language and domain gap, and evaluate it on dataset A:

- train the model on dataset A;
- train the model on dataset A translated to language L_B: language gap;
- train the model on dataset B translated to language L_A: domain gap;
- train the model on dataset B: domain gap and language gap.

We report in table 6.2 a few results using the Dachs EN, Dachs FR and Germeval datasets.

Domain gap	No		Yes	
Language gap	No	Yes	No	Yes
Training data	Dachs FR	Dachs FR → EN	Dachs EN → FR	Dachs EN
Test on Dachs FR	75.0 (± 1.4)	67.1 (± 0.6)	59.3 (± 2.6)	51.9 (± 0.8)
Training data	Dachs EN	Dachs EN → FR	Dachs FR → EN	Dachs FR
Test on Dachs EN	74.2 (± 0.2)	68.3 (± 0.0)	62.5 (± 0.4)	61.8 (± 0.6)
Training data	Germeval	Germeval → EN	Dachs EN → DE	Dachs EN
Test on Germeval	61.6 (± 1.9)	57.6 (± 1.6)	40.6 (± 0.3)	39.0 (± 1.5)
Training data	Dachs EN	Dachs EN → DE	Germeval → EN	Germeval
Test on Dachs EN	61.6 (± 1.9)	68.6 (± 1.2)	47.7 (± 1.0)	46.7 (± 0.7)

Table 6.2. Macro-F1 scores when training a XLM-R+LASER model with different datasets, creating domain and/or language gaps.

Two conclusions come from these results. We first see that even though we use a multilingual model, translations have some influence. In particular when the training dataset is different from the test dataset, it is useful to translate the training data into the test language (see the last two columns of table 6.2). It is better to have no language gap, even if this means adding some noise during translations).

Moreover, as stated previously, the domain gap is clearly more problematic than the language gap. The same pattern applies for all examples, with a drop of performance due to the domain gap at least twice as important as the drop of performance due to the language gap. It shows that *cross-domain* classification is actually harder than *cross-lingual* classification, and meticulous care should be given to gather a good selection of compatible datasets. This also means that in some cases training data from other languages might be better than training data in the target language and should not be excluded.

6.3 | Tuning the models for a target application

Until now, we have used the macro-F1 score to measure the general performance of our models, for the reasons cited in section 3.2. It accords the same importance to all classes, and to the recall and precision of those classes, but depending on the nature of the problem we might want to give more importance to the recall or precision of some classes. This is often the case with hate speech detection, where it is usually more important to have a high recall on the `hate` class rather than a high precision.

One solution would be to measure only the recall on `hate`, but then it is not possible to detect extreme cases where the model does not learn anything (see in section 3.2, if we have a model that always predict `hate` we will have a score of 100%). We wanted a metric that could offer a general overview of the performance of a model while giving more importance to the recall on `hate` and `offensive`, and that can still detect and avoid extreme cases. We define a custom metric that we call *F-hate*, which can be used to optimise a given model for a specific application, with a certain degree of severity required to block toxic content. This is a variant of the F1-score, with weights given to the precision and recall. We define

$$\text{F-hate}_{none} = (1+\beta^2) * \frac{\text{precision}_{none} * \text{recall}_{none}}{\text{precision}_{none} + \beta^2 * \text{recall}_{none}}$$

$$\text{F-hate}_{offensive} = (1+\beta^2) * \frac{\text{precision}_{offensive} * \text{recall}_{offensive}}{\beta^2 * \text{precision}_{offensive} + \text{recall}_{offensive}}$$

$$\text{F-hate}_{hate} = (1+\beta^2) * \frac{\text{precision}_{hate} * \text{recall}_{hate}}{\beta^2 * \text{precision}_{hate} + \text{recall}_{hate}}$$

where, for $\beta > 1$, we give on the `none` class more importance to the precision than to the recall (when we mark a message as `none` we want to be sure that it is indeed `none`), and on the `offensive` and `hate` classes we give more importance to the recall (we do not want to miss toxic content). From this we can derive the global F-hate metric, defined as

$$\text{F-hate} = \frac{\text{F-hate}_{hate} + \text{F-hate}_{offensive} + \text{F-hate}_{none}}{3}$$

If we take for instance $\beta = 2$, it means that the recall on `hate` is twice as important as the precision on `hate`, the recall on `offensive` is twice as important as the precision on `offensive`, and the precision on `none` is twice as important as the recall on `none`. This would be a system where we do not want to let toxic messages pass, and we prefer to have to manually white-list acceptable content. Increasing β will make this system more strict. If we take $\beta = 1$, this corresponds to the macro-F1 score, where recall and precision have the same importance.

Choosing $\beta > 1$ is usually the best choice since toxic comment classification cannot be considered a symmetrical problem. It naturally depends on the final application, but most of the time one would rather have a system where no toxic content is missed than a system were toxic content easily gets through, even if there are a few false positives.

With $\beta < 1$, it would create a system that tries not to block any normal content, even if it means letting toxic content through more often. This case is probably less common, but the results are still interesting and might be applicable to some specific applications (for instance to avoid being accused of censoring free-speech it is necessary to let all non-toxic content through).

More generally, this method of creating a custom metric based on the macro-F1 can be used on other classification tasks, and conclusions drawn in this section should be applicable to other contexts.

With this metric defined we can now train models, but instead of using the macro-F1 as a metric for the early stopping we use the F-hate metric with $\beta > 1$, which should make the model stop training at a point with a high recall on `hate` and `offensive`. We look at results for the Dachs EN dataset in table 6.3, with a simple transformer model, RoBERTa [15] with the CNN classification head (section 4.2.1), once using the macro-F1 score as the early stopping metric, and once using the F-hate metric with $\beta = 2$. We also report results using the weighted cross-entropy loss (WCEL) instead of the cross-entropy loss (CEL) (see section 4.3.1), as it causes a similar effect to the use of the F-hate.

				None		Offensive		Hate	
Loss	Metric	macro-F1	F-hate	P	R	P	R	P	R
CEL	macro-F1	**73.5**	72.4	85.8	**88.2**	**68.9**	64.6	**69.1**	65.0
	F-hate	70.8	74.6	88.6	80.2	61.6	72.0	55.0	71.7
WCEL	macro-F1	72.5	72.8	86.5	85.8	66.2	66.8	64.9	65.8
	F-hate	70.3	**76.8**	**90.5**	75.8	58.9	**76.3**	51.1	**79.5**

Table 6.3. Results depending on the early stopping metric and the loss function on Dachs EN, using RoBERTa with the CNN classification head. The F-hate metric uses $\beta = 2$.

With the original version (macro-F1 and CEL), recall and precision are relatively similar. The use of the F-hate metric increases the recall on `hate` and `offensive` by 6-8%, and the precision on `none` increases by 3%. To compensate, the three other metrics, recall on `none` and precision on `hate` and `offensive` decrease. Going from the unweighted CEL with the macro-F1 to the WCEL with the macro-F1 has a similar effect but to a lesser extend, while using the WCEL with the F-hate increases this effect even more. If we perform the same experiment with Dachs FR, we obtain very similar results (see table 6.4). To accord more importance to the recall of the toxic classes, we should use the WCEL with the F-hate metric. The CEL with the macro-F1 is at the opposite, with the lowest recall on toxic classes. The two other options (CEL with F-hate and WCEL with macro-F1) once again find themselves in between the two extremes.

				None		Offensive		Hate	
Loss	Metric	macro-F1	F-hate	P	R	P	R	P	R
CEL	macro-F1	**79.3**	77.9	89.7	**92.7**	**75.0**	67.0	**79.3**	74.5
	F-hate	78.5	80.2	91.5	89.5	70.0	71.4	68.7	81.7
WCEL	macro-F1	77.8	80.7	92.7	86.3	65.0	76.8	69.5	78.8
	F-hate	75.2	**81.2**	**94.1**	80.6	59.1	**80.0**	61.7	**83.6**

Table 6.4. Effects when changing the early stopping metric and the loss function, on Dachs FR, using FlauBERT with the CNN classification head. The F-hate metric uses $\beta = 2$.

If the recall on `hate` was used a the sole metric, we would see something similar to the score obtained with the WCEL and the F-hate metric, with a recall on `hate` significantly higher than the precision on `hate`. However, this difference would be larger, and we might lose a lot of precision for only an insignificant improvement in recall. For instance, a model with $recall_{hate} = 85\%$ and $precision_{hate} = 40\%$ would be chosen over a model with $recall_{hate} = 84\%$ and $precision_{hate} = 70\%$. This is problematic, as we do not want to sacrifice too much of any

metric. We also observed some cases where the initial prediction of the model was that all content was `hate`, due to the random initialisation of the network weights, and if we tracked only the recall on `hate` the training would quickly stop as we would already have reached a maximal value of the metric.

It is straightforward to change the *F-hate* score with the desired ratio to give more importance to some classes. This choice is mostly arbitrarily, and one would likely need to test a few values of β to find a good compromise. We show in figure 6.1 an experiment on Dachs EN (RoBERTa with the CNN classification head) with the CEL, where we change the value of β in the F-hate metrics. We plot for $\beta \in \{1/3, 1/2, 2/3, 1, 3/2, 2, 3\}$ the metrics computed from the average of three runs over random test sets. We also report the standard error for each value, computed as $\frac{\sigma}{\sqrt{3}}$, where σ is the standard deviation.

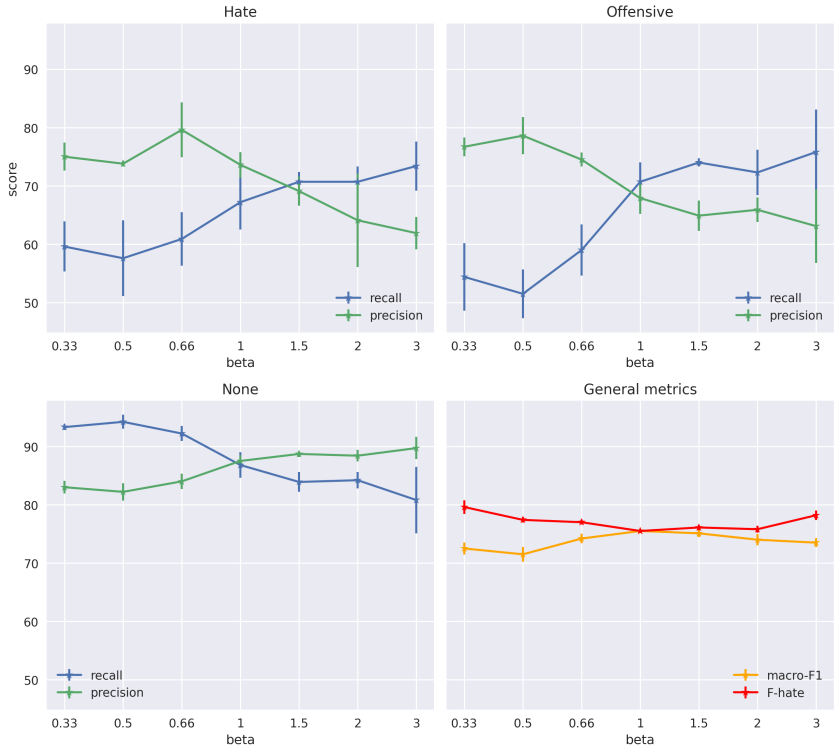

Figure 6.1. Evolution of the recall, precision and macro metrics on Dachs EN, depending on the β value in the F-hate metric, using RoBERTa with a CNN classification head.

When we increase the β coefficient, the recall on `hate` and `offensive` and the precision on `none` increase, while the precision on `hate` and `offensive` and the recall on `none` decrease, and the opposite happens when decreasing the β coefficient. This is a symmetrical problem, with $\beta = 2$ having the opposite effect of $\beta = 1/2$. When $\beta = 1$, the macro-F1 is the same as the *F-hate*. The further from 1 this value is, the lower the macro-F1 gets, and the higher the *F-hate* gets.

The previous results show that the *F-hate* is useful to tune the model to a desired level of severity in a monolingual context, but we still need to see if the same conclusions apply to a multilingual context. We experimented with XLM-R [24] with the CNN head (section 5.1.1) joined with the LASER+MLP block (section 5.2.3). We trained the model on Jigsaw and Dachs FR, and evaluated it on Germeval, Dachs ES, Dachs GR and the Indonesian dataset. We trained two versions on this model, one with the macro-F1 used as the early stopping and once with the F-hate with $\beta = 2$ (twice as much importance to recall on `hate` and `offensive` and to precision on `none`).

Going from the macro-F1 to the F-hate, we were able to observe the expected effect on Jigsaw and Dachs FR, the training languages. The pattern was clear, and in both cases the recall on `hate` and `offensive` increased and the precision on `none` also increased. However, with the other datasets that were evaluated in the zero-shot context we did not consistently observe this effect. For some classes and datasets it was present, but for others not, and we could not identify any pattern that might explain the results. Since the F-hate does not actually change the architecture but only stops the training at a point with optimal scores on the validations dataset, we would need the datasets in the zero-shot languages to be extremely close to the early stopping dataset to allow for a transfer of this effect. In practice it is often not the case, and the F-hate does not properly apply to zero-shot classification.

6.4 | Multilingual model or multiple monolingual models

As seen in the table 5.8, when multiple languages have available training data, there is not one unique model that performs optimally on all languages, and for each language a different architecture would be better suited. From these observations, we propose in figure 6.2 a pipeline that could be beneficial for multilingual classification.

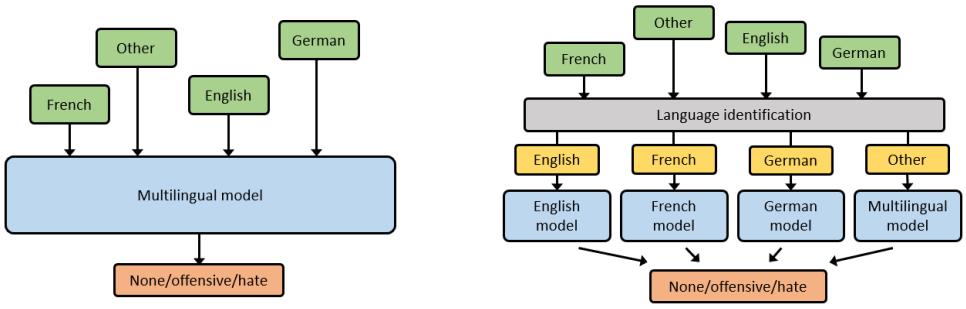

Figure 6.2. On the left, the original pipeline with a single model to classify data from any language. One the right, the advanced pipeline where we suppose French, English and German training data. Data in any other language falls in the *other* category.

Given a set S of languages for which we have some training data, we create $|S|$ individual monolingual classifiers, one for each language in S, and a language identification module. One multilingual classifier is also trained using any of the architectures presented in chapter 5. When presented with a new sentence to classify, if it is written in one of the languages in S the architecture sends the sentence to the associated monolingual classifier, otherwise it goes to the multilingual classifier. This gives us scores on known languages as good as possible using monolingual models, while still being able to classify any other language. There is a trade-off between performance and cost; using a single multilingual model for all languages will require less memory but will have lower scores; whereas for a more advanced pipeline more memory will be required but the scores will be significantly better.

The final choice of architecture also comes down to the data the model will be applied to. If for instance most of the expected data is in English, with only a few messages in French or German, a model with good performance in English (equivalent to a monolingual English model) but slightly worse on other languages can be used. On average, the performance will still be better than with a purely multilingual model, and adding monolingual models for French and German might not be worth the added cost.

6.5 | Model selection

Due to the chosen training methodology and inherent variance when training transformers, all of our results have quite a lot of variance, which often makes the comparison between multiple models complex. In some cases it is still possible to say that an architecture is better than another, but we expect that one might get different results when trying to reproduce some of our experiments. Overall, we think we were able to demonstrate the potential of advanced classification heads, and that for cross-lingual classification the joint models are useful and translating data into a single language to use monolingual models can work, depending on the targeted languages. The largest variable here remains the choice of datasets. Depending on the available training data and the desired target language and evaluation dataset, some architectures will be more adapted than others.

We therefore cannot confidently select a best architecture. For monolingual models, we can confidently say that RoBERTa [15], FlauBERT [38] and XLM-RoBERTa [24] are the best transformers for our English, French and German datasets, but we are less confident with the other architectural changes. For cross-lingual classification, with our datasets and with a specific subset of languages, we found that the joint RoBERTa+LASER model with translations (section 5.2.3) was on average better (table 5.8 shows the usefulness of translations and joint models in at least some cases). The XLM-R+LASER model was also competitive, without requiring translations, although the translation-based models are still overall better than the purely multilingual models with our datasets.

When tasked with developing a multilingual model based on the present work, one should evaluate the various models for all target languages to determine the best choice (and this choice should also depend on the expected proportion of data from each language). Ideally, this evaluation data should not originate from the datasets mentioned in this project but should be gathered specifically for the task at hand. Additionally, the different options presented in the first chapter (in section 4.2) should also be evaluated for cross-lingual models.

Finally, this choice will also come down to a compromise between performance and cost. Some of the evaluated options and architectures require significantly more computation power, and a consequent amount of time would need to be dedicated to finding the optimal model. This investment might not be worth for potentially only a few percents of improvements in the final score compared to a simple base transformer model.

7 | Conclusions and Future work

7.1 | Conclusions

In this thesis we tackled monolingual and cross-lingual classification. In both cases, we evaluated existing methods on multiple available datasets, and then proposed various approaches to improve the performance of these methods.

With monolingual classification, we showed that the recently introduced transformer-based models clearly outperform classical methods, and that we can improve results slightly by using different classification heads. We also showed the difficulty of combining multiple datasets; using a multitask approach is usually better than naively merging datasets, but the domain gap between them is usually too wide to lead to any improvement. Other methods were less successful, but the high standard deviations of results make it hard to make confident conclusions so we cannot exclude the possibility that these methods might work with other datasets.

With cross-lingual classification, we used data translation to introduce multiple joint-learning architectures. We combined transformer-based models with LASER embeddings and the Hurtlex representation, leading to better results than the transformer models by themselves. These approaches were evaluated on multiple languages, including languages for which we had no training data. The translation-based models with joint learning usually yield slightly better results than purely multilingual models, but come with the added translation cost. Regrettably, we were not able to create one single multilingual model that performed as good on English, French and German as the monolingual models trained on each language individually, but we still reached relatively close results considering the difficulties introduced when merging datasets.

We also proposed a custom metric, the *F-hate*, that can be adapted and used to give more importance to the precision or the recall on some classes. Using this metric for early stopping, it is possible to create models with varying levels of severity in their classification, and it is therefore relatively easy to adapt them to specific target applications with various levels of severity desired. We were not able to transfer this method to a zero-shot context, in part due to the domain gap.

The domain gap seems to be the largest challenge that needs to be overcome before being able to create performant multilingual models. This effect is already important when comparing multiple datasets in the same language and is sometimes even more present across languages due to large differences in societal and cultural issues between populations. We showed that the domain gap is in fact a larger problem than the language gap to implement cross-lingual models. The datasets chosen for any multilingual problem need to most importantly have a low domain gap between them if we want decent zero-shot classification scores, and the difference of languages is not as problematic.

7.2 | Future Work

There are still a lot of problems with the methods presented in this thesis that lead to scores that are often not good enough for a sensitive task like hate speech detection. We list here some leads that we did not have time to investigate but might improve the models.

Multitask-learning for cross-lingual data: In chapter 4, we presented two approaches to data combination: data merging and multitask learning. However, in the second part on cross-lingual classification (chapter 5), we only used data merging to combine data from multiple languages. We think there is some potential in combining data using multitask learning, having for instance specific classification heads for each language in the training data, and one classification head used for the other unknown languages, which could partially alleviate the domain gap.

LASER fine-tuning: With the transformer models (section 4.1.4), we perform fine-tuning with our training data to improve the quality of the language model representation inside the transformer. However, with LASER embeddings (section 5.1.1), we use a frozen version of the encoder network to generate the embeddings. Performing fine-tuning on this encoder network might lead to sentence embeddings more adapted to toxic content, and better classification scores.

Other transformer models: New transformer models are regularly introduced, with for instance T5 [56], ELECTRA [57] or LongFormer [58]. All these models improve in some ways the models used in this project and are all available in the transformers library[1] for easy use and combination with a classification head. There are also some older transformer models that we did not try, such as GPT-2 [59] or BART [60]. It would be interesting to evaluate their performance on our task.

Continuous model tuning: It was discussed that the use of language changes over time, with new expressions and insults added over time, and changes to the meaning of some words. The models trained for this project might perform well when applied to data gathered recently, but they might not be relevant anymore in a few years. This would require training a new model every few years, with up-to-date data to keep up with the evolution of language, which can get expensive. Instead, it might be possible to re-use an existing model and adapt it to new expressions, trends and topics. One could for instance remove the classification head from a trained model, while keeping the transformer itself, and then train a new model using a fresh classification head. We could then preserve all the useful information in the transformer regarding the language model, adapted to toxic content, while having a classification head more adapted to current data. This additional data could be gathered over time from manually reported messages, and allow for continuous adaptation of models to new data, without requiring a completely new training.

Bridging the domain gap: The largest challenge with cross-lingual classification is the domain gap, and the discovery of methods to reduce it would probably lead to a significant improvement in the performance of current models. As a temporary solution, we could find techniques to check the compatibly of datasets. Semi-supervised clustering could for instance be used to properly detect the themes present in multiple datasets, and we could then select only subsets of toxic content present in all datasets for better data compatibility. Depending on the target task, one could automatically select the adapted toxic training data; the domain gap would then be less problematic which could improve the quality of the classifier.

[1] List of supported architectures on https://huggingface.co/transformers/model_summary.html

References

[1] Polychronis Charitidis et al. *Towards countering hate speech and personal attack in social media*. 2019. arXiv: 1912.04106v1 [cs.IR].

[2] Thomas Davidson et al. *Automated Hate Speech Detection and the Problem of Offensive Language*. 2017. arXiv: 1703.04009 [cs.CL].

[3] Nedjma Ousidhoum et al. *Multilingual and Multi-Aspect Hate Speech Analysis*. 2019. arXiv: 1908.11049 [cs.CL].

[4] Betty van Aken et al. "Challenges for Toxic Comment Classification: An In-Depth Error Analysis". In: *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 33–42. DOI: 10.18653/v1/W18-5105. URL: https://www.aclweb.org/anthology/W18-5105.

[5] Zeerak Waseem and Dirk Hovy. "Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter". In: *Proceedings of the NAACL Student Research Workshop*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 88–93. DOI: 10.18653/v1/N16-2013. URL: https://www.aclweb.org/anthology/N16-2013.

[6] Pinkesh Badjatiya et al. "Deep Learning for Hate Speech Detection in Tweets". In: *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion* (2017). DOI: 10.1145/3041021.3054223. URL: http://dx.doi.org/10.1145/3041021.3054223.

[7] Z. Zhang, D. Robinson, and J. Tepper. *Detecting hate speech on Twitter using a convolution-GRU based deep neural network*. Ed. by A. Gangemi et al. © 2018 Springer International Publishing AG, part of Springer Nature. This is an author produced version of a paper subsequently published in Lecture Notes in Computer Science. Uploaded in accordance with the publisher's self-archiving policy. June 2018. URL: http://eprints.whiterose.ac.uk/128405/.

[8] Björn Gambäck and Utpal Kumar Sikdar. "Using Convolutional Neural Networks to Classify Hate-Speech". In: *Proceedings of the First Workshop on Abusive Language Online*. Vancouver, BC, Canada: Association for Computational Linguistics, Aug. 2017, pp. 85–90. DOI: 10.18653/v1/W17-3013. URL: https://www.aclweb.org/anthology/W17-3013.

[9] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[11] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. 2016. arXiv: 1607.04606 [cs.CL].

[12] Matthew E. Peters et al. "Deep contextualized word representations". In: *Proc. of NAACL*. 2018.

[13] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805 [cs.CL].

[14] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[15] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[16] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. arXiv: 1906.08237 [cs.CL].

[17] Marzieh Mozafari, Reza Farahbakhsh, and Noel Crespi. *A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media*. 2019. arXiv: 1910.12574 [cs.SI].

[18] Chi Sun et al. *How to Fine-Tune BERT for Text Classification?* 2019. arXiv: 1905.05583 [cs.CL].

[19] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: (2020). arXiv: 2005.14165 [cs.CL].

[20] Sai Saketh Aluru et al. *Deep Learning Models for Multilingual Hate Speech Detection*. 2020. arXiv: 2004.06465 [cs.SI].

[21] Xiaojun Wan. "Co-Training for Cross-Lingual Sentiment Classification". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, Aug. 2009, pp. 235–243. URL: https://www.aclweb.org/anthology/P09-1027.

[22] Endang Wahyu Pamungkas and Viviana Patti. "Cross-domain and Cross-lingual Abusive Language Detection: A Hybrid Approach with Deep Learning and a Multilingual Lexicon". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 363–370. DOI: 10.18653/v1/P19-2051. URL: https://www.aclweb.org/anthology/P19-2051.

[23] Mikel Artetxe and Holger Schwenk. *Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond*. 2018. arXiv: 1812.10464 [cs.CL].

[24] Alexis Conneau et al. *Unsupervised Cross-lingual Representation Learning at Scale*. 2019. arXiv: 1911.02116 [cs.CL].

[25] Guillaume Lample and Alexis Conneau. "Cross-lingual Language Model Pretraining". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[26] Antigoni-Maria Founta et al. *Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior*. 2018. arXiv: 1802.00393 [cs.SI].

[27] 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*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 46–57. DOI: 10.18653/v1/W19-3506. URL: https://www.aclweb.org/anthology/W19-3506.

[28] Ellery Wulczyn, Nithum Thain, and Lucas Dixon. *Ex Machina: Personal Attacks Seen at Scale*. 2016. arXiv: 1610.08914 [cs.CL].

[29] Ona de Gibert et al. "Hate Speech Dataset from a White Supremacy Forum". In: *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 11–20. DOI: 10.18653/v1/W18-5102. URL: https://www.aclweb.org/anthology/W18-5102.

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://www.aclweb.org/anthology/P16-1162.

[32] Yonghui Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[33] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[34] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[35] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771 (2019).

[36] Yukun Zhu et al. "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (Dec. 2015). DOI: 10.1109/iccv.2015.11. URL: http://dx.doi.org/10.1109/iccv.2015.11.

[37] Zihang Dai et al. *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*. 2019. arXiv: 1901.02860 [cs.LG].

[38] Hang Le et al. *FlauBERT: Unsupervised Language Model Pre-training for French*. 2019. arXiv: 1912.05372 [cs.CL].

[39] Louis Martin et al. *CamemBERT: a Tasty French Language Model*. 2019. arXiv: 1911.03894 [cs.CL].

[40] Jaejun Lee, Raphael Tang, and Jimmy Lin. *What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning*. 2019. arXiv: 1911.03090 [cs.CL].

[41] Marian-Andrei Rizoiu et al. *Transfer Learning for Hate Speech Detection in Social Media*. 2019. arXiv: 1906.03829 [cs.SI].

[42] N. V. Chawla et al. *SMOTE: Synthetic Minority Over-sampling Technique*. 2011. arXiv: 1106.1813 [cs.AI].

[43] Jason Wei and Kai Zou. *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks*. 2019. arXiv: 1901.11196 [cs.CL].

[44] Francis Bond and Kyonghee Paik. "A Survey of WordNets and their Licenses". In: *Proceedings of the 6th Global WordNet Conference (GWC 2012)*. 64–71. Matsue, 2012.

[45] Benoît Sagot and Darja Fišer. "Building a free French wordnet from multilingual resources". In: *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*. Ed. by European Language Resources Association (ELRA). Marrakech, Morocco, 2008.

[46] Christiane Fellbaum, ed. Cambridge, MA: MIT Press, 1998.

[47] Rob van der Goot and Gertjan van Noord. *MoNoise: Modeling Noise Using a Modular Normalization System*. 2017. arXiv: 1710.03476 [cs.CL].

[48] Steven Zimmerman, Udo Kruschwitz, and Chris Fox. "Improving Hate Speech Detection with Deep Learning Ensembles". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://www.aclweb.org/anthology/L18-1404.

[49] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[50] Alexis Conneau et al. "Word Translation Without Parallel Data". In: *arXiv preprint arXiv:1710.04087* (2017).

[51] Philip Gage. "A new algorithm for data compression". In: ().

[52] Marcin Junczys-Dowmunt et al. "Marian: Fast Neural Machine Translation in C++". In: *Proceedings of ACL 2018, System Demonstrations*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 116–121. URL: http://www.aclweb.org/anthology/P18-4020.

[53] Jörg Tiedemann. "Parallel Data, Tools and Interfaces in OPUS". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2214–2218. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[54] Jörg Tiedemann and Santhosh Thottingal. "OPUS-MT — Building open translation services for the World". In: *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*. Lisbon, Portugal, 2020.

[55] Viviana Patti Elisa Bassignana Valerio Basile. "Hurtlex: A Multilingual Lexicon of Words to Hurt". In: *In Proceedings of the Fifth Italian Conference on Computational Linguistics (CLiC-It 2018)*. 2018.

[56] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. arXiv: 1910.10683 [cs.LG].

[57] Kevin Clark et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. arXiv: 2003.10555 [cs.CL].

[58] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL].

[59] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019.

[60] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].