Human Language Technologies

Automatic Misogyny Identification

Grieco Giuseppe
g.grieco6@studenti.unipi.it

Lepri Marco
m.lepri2@studenti.unipi.it

Sangermano Mattia
m.sangermano1@studenti.unipi.it

Academic Year: 2019/2020

Contents

1 Introduction

Unfortunately nowadays more and more episodes of toxic behaviour can be found on social medias, often directed to women through harassment or misogynistic comments, where the misogynists hide themselves behind the security of the anonymity. Given the huge amount of data produced every second on social media platforms, it is crucial to automatically identify this kind of comments and behaviour and remove them *before* they are publicly shown. However, it is important to guarantee the fairness of the identification system in order to avoid discrimination or prejudices towards a certain group of people.
Therefore, the task addressed here is the one of Automatic Misogyny Identification (AMI), introduced in the context of the 2020 EVALITA competition[1]. The AMI task includes two subtasks:

- **Subtask A - Misogyny and Aggressive Behaviour Identification**: recognize if a text is misogynous or not and, in case of misogyny, if it expresses an aggressive attitude.
- **Subtask B - Unbiased Misogyny Identification**: discriminate misogynistic contents from the non-misogynistic ones, while guaranteeing the fairness of the model (in terms of unintended bias) on a synthetic dataset.

The two tasks come with an unique dataset of tweets manually labeled according to two levels: *Misogyny* and *Aggressiveness* both taking 0 or 1 as value. For the subtask B, also a synthetic dataset is provided, labeled only according to *Misogyny*. Table 1 shows some example tweets from the two datasets. For the synthetic dataset, a set of *identity terms* is defined on which the unintended bias is computed ([1],[2],[3]). The identity terms are placed in synthetic templates that convey misogynistic and non misogynistic messages. The competition also allow the usage of external datasets for the training, however distinguishing the submitted solution based on whether external data has been used or not.

	Text	**Misogynous**	**Aggressive**
Raw	Stupida troia! Meriti di morire	1	1
	Le ragazze sono tutte puttane	1	0
	@chiellini sei una cagna	0	0
Synthetic	*donne* dovrebbero essere protette	0	
	ragazze dovrebbero essere protette	0	
	donne dovrebbero essere uccise	1	
	ragazze dovrebbero essere uccise	1	

Table 1: Example of tweets from the raw and the synthetic datasets

[1] `https://amievalita2020.github.io`

Our work focused only on the subtask A, without using any external dataset to enhance the models. Therefore, only the raw dataset is analyzed and used for the experiments.

2 Data analysis

The dataset is made of 5000 tweet examples, each associated with two target columns: one for the misogyny and one for the aggressiveness. The two columns are not independent since each example can be labelled as aggressive only if it is also labelled as misogynous. Hence, there are only three classes possible: `Non-Misogynous` (both columns are 0), `Misogynous` (only `misogyny` column is 1) and `Aggressive` (both columns are 1).
In the following, we analyze the dataset after applying a basic tokenization and normalization. In particular, characters were lower-cased, the punctuation and the special characters were removed and the words were separated in correspondence of white spaces.

2.1 General Statistics

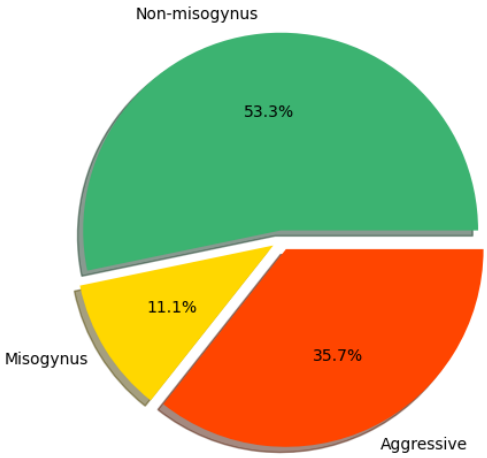

Figure 1: Classes distribution

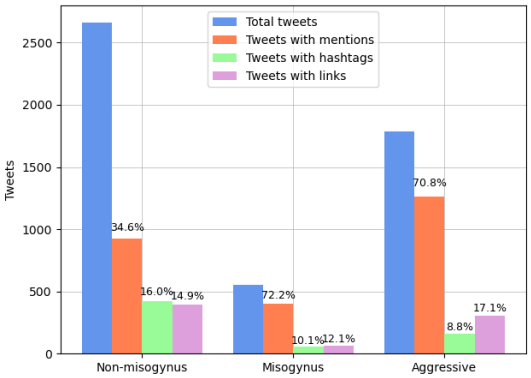

Figure 2: Content statistics

Figure 1 shows the distribution of the three classes in the dataset. The `misogyny` column is fairly balanced, with 53.3% of the tweets labelled with a 0 and 46.7% of the tweets labelled with a 1. The `aggressiveness` columns is more unbalanced, with just 1/3 of the tweets being labelled with a 1. However, considering the three classes independently, the dataset is way more unbalanced, with just 11% of the tweets labelled as `Misogynous` while 53.3% of the dataset labelled as `Non-Misogynous`.
Figure 2 shows statistics about mentions, hashtags and links inside the tweets, divided by class. Mentions seem to appear in most of the `Misogynous` and `Aggressive` tweets, while around 1/3 of the `Non-Misogynous` tweets contain them. Hashtags and links appear

less frequently and are better distributed among the classes, although `Non-Misogynous` tweets are more likely to contain either of them.

2.2 Length Statistics

Figures 3 and 4 show the tweet lengths distribution for the whole dataset and for the single classes respectively. Concerning the general statistics both the length in characters and in words follow a skewed normal distribution with the majority of the tweets being short (less than 20 words). There are also few hundreds of tweets whose length in words is greater than 40 which are, however, not equally distributed among the classes. Indeed, `Non-Misogynous` tweets have a higher average length, both in terms of characters and in words, with many tweets being longer than 40 words. On the other side, `Misogynous` and `Aggressive` tweets tend to be shorter, with the `Aggressive` ones being slightly longer on average, while few tweets exceeds 40 words in length. In all the cases, the tweets barely exceed 50 words in length with only 43 tweets being longer, most of which labelled as `Non-Misogynous`.

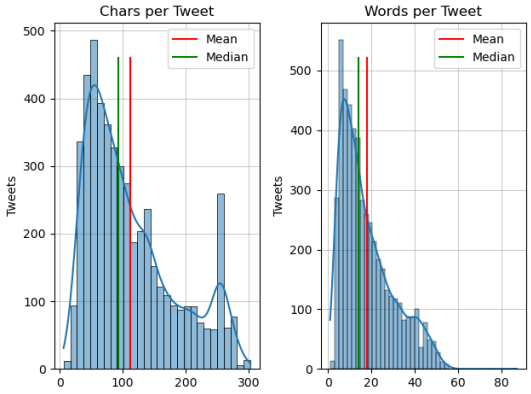

Figure 3: Global Length Distribution

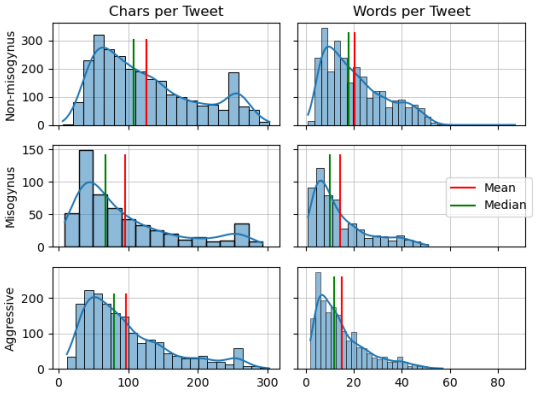

Figure 4: Per-class Length Distribution

2.3 Words Statistics

Figure 5 shows the distribution of the words frequency. Overall, the dataset contains 13280 distinct words, without considering mentions and links. Of these, two thirds appear only once in the dataset, while just 13% of them appear two times. Of the remaining words, the 5.7% appear more than 10 times, with many of the highest frequency words being stopwords. Figure 6 shows the 20 most frequent words, excluding the stopwords. Just a couple of them exceed 500 occurrences, while the majority has a frequency around 200 occurrences. Many of these are insults of offensive terms, especially in the top ranked terms, and the frequency is rarely distributed accordingly with the classes distribution. Indeed the majority of the words either appear mostly in `Non-Misogynous` tweets or appear mostly in the other classes of tweets.

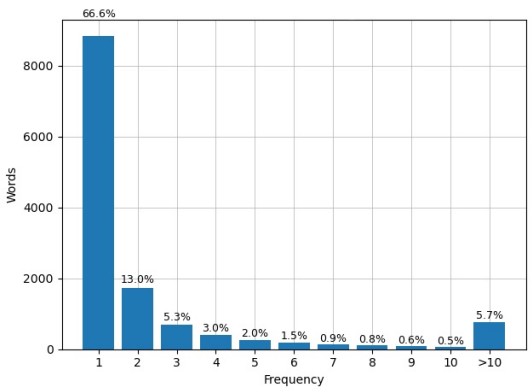

Figure 5: Word Frequency Distribution

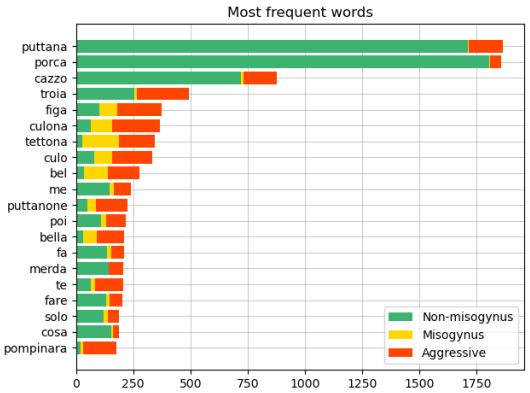

Figure 6: Most frequent words

Concerning Out-of-Vocabulary (See Section 3.5 for which vocabulary was used) words, Figure 7 shows the percentage and the total frequency (in percentage) of OOV words. Despite being around 1/6 of the total words, their total occurrences are less than the 4% of the total words, meaning that most of them appear with low frequency and might be typos.

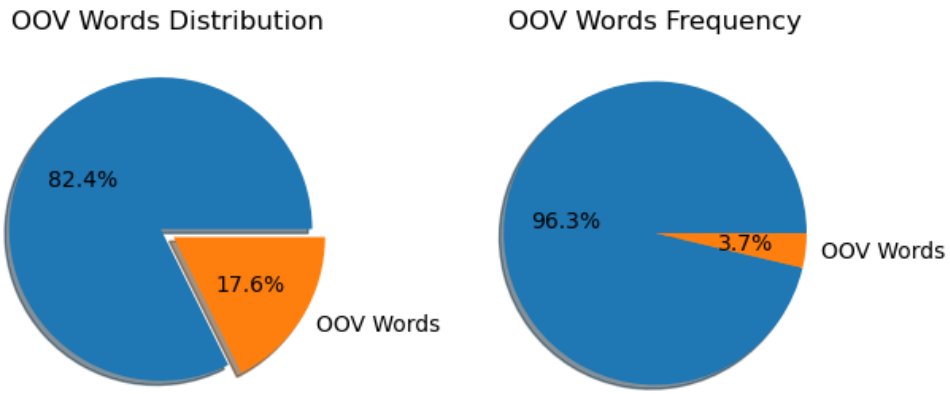

Figure 7: Out-of-Vocabulary words statistics

3 Preprocessing

The preprocessing routine can be divided into 5 sub routines shown in their execution order in figure 8.

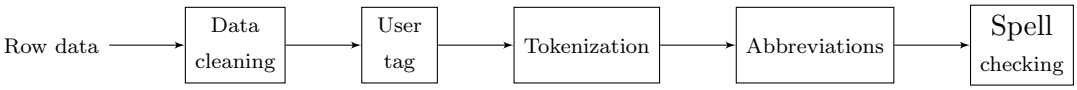

Figure 8: High-level view of the operations carried out as preprocessing

3.1 Data cleaning

In order to clean the data from unusable characters or words we performed 3 steps in the following execution order:

1. Remove all the links since it does not provide any additional information
2. Limit the repeating letter to 2: e.g. "ciaooooo" is transformed into "ciaoo"
3. Remove all special characters except ones belongs to the following set: {`@`, `£`, `+`, \, `-`, `=`, `€`, `$`, `%`, `_`}. Note that the symbol `#` is removed so that hashtags are treated as normal words in the tweets. This is done since often hashtags are used as part of the sentence and not just as additional information after the tweet

3.2 User tag processing

Since the user tags are more common in `Misogynous` and `Aggressive` tweets, we do not use a special token, e.g. <`tag`>, in their place. Instead, we replace the tags with the gender of the corresponding user in order to know whether the tweet is referring to a man or a woman. To do so, we first identify the user tag through the `Twitter API` that returns the user's real name, associated with the profile. This is used to obtain the gender thanks to `genderize.io` which, given a real name, returns the probabilities that the name is a male or female name. In case the name is identified as male, the tag is replaced with the word *Uomo*. In case the name is identified as female, the tag is replaced with the word *Donna*. In case the name is not available (often due to privacy settings of twitter's user) or it is not possible to identify the gender of the name, the tag is replaced with the word *Persona*.

3.3 Tokenization

The tokenization is made using `nltk TweetTokenizer` which is a tokenizer made specifically to deal with tweets' usual typing, which differs from normal text.
Before applying the tokenization all the characters are lowercased, so that the final dictionary is smaller, although this could reduce the expressiveness of the tweets (e.g. shouting)

3.4 Abbreviations processing

Given that tweets usually are written using slang language and abbreviations, the tweets pass through a standardization phase. In this, the abbreviations are replaced by the corresponding full word or expression.

3.5 Spell checking

In order to further standardize the tweets and improve the general quality of the dataset, the tweets pass through a spell checking phase, in order to remove typos or other kind of errors.
The spell checker used (`PySpellChecker`) is based on the well known Levenshtein Distance algorithm. It looks for all the permutations, within an edit distance of 2, of a given word. Then it compares them with the words in a dictionary, selecting as correction the word with higher frequency among those found. In our case, the dictionary used is the merge between AlBERTo[4] dictionary and NLPL dictionary[5].

Furthermore, we also try to reconstruct unrecognized words in case of typos where a space is inserted between the letters of the same word. In particular, we check for words written with all the letters separated by a space (e.g. `A V A N T I`), often used to add emphasis to the tweet, and compact the word together. We also check for two subsequent words which do not match any word in the dictionary and try to concatenate them and see if the resulting word matches any one in the dictionary. These two operations are performed together with abbreviation processing and spell checking.

3.6 Dataset Analisys

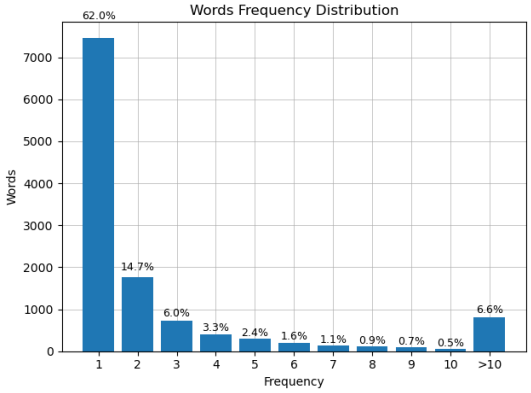

Figure 9: Postprocessing word frequency distribution

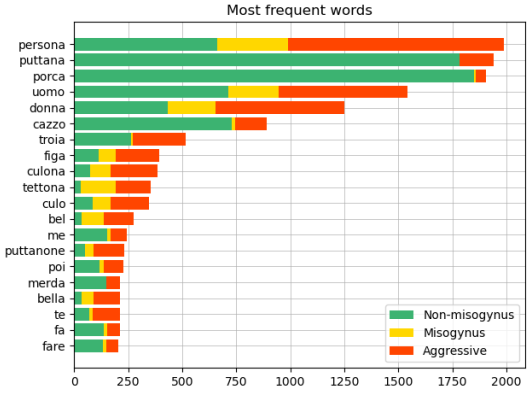

Figure 10: Post processing most frequent words

Figure 9 shows the words frequency distribution after the preprocessing. The number of distinct words is decreased of more than 1000 words, for a total number of distinct

words of 12046. Among these, still around 1/3 appear only once in the dataset, while the number of words appearing more than 10 times is increased to the 6.6%. Figure 10 shows again the 20 most frequent words excluding the stopwords. *Persona* is now the most frequent word, due to the tag processing of the mentioned users. Also *Uomo* and *Donna* are now part of the most frequent words, still as a result of the previous process. Figure 11 shows the distribution of the processed tweets length, in words. The tweets are still shorter than 20 words on average for `Misogynous` and `Aggressive` tweets, while `Non-Misogynous` tweets are still longer on average with many tweets exceeding 40 words of length.

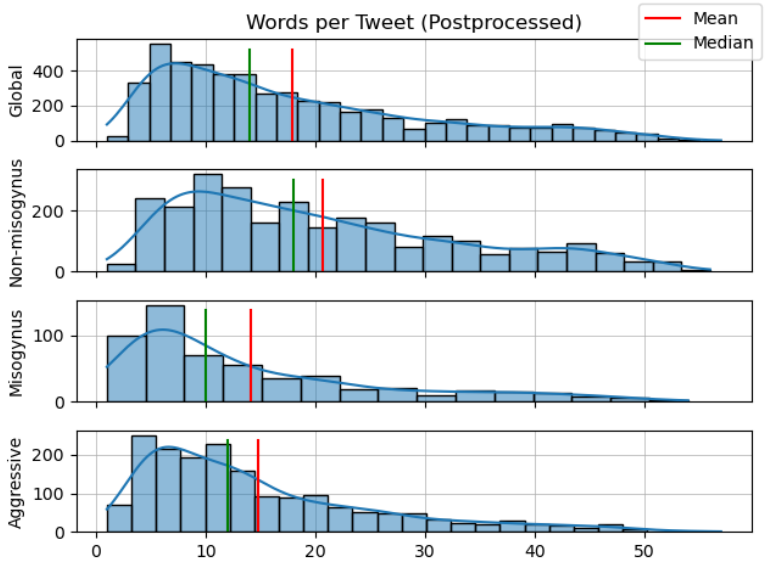

Figure 11: Postprocessing length statistics

For this reason we decided to truncate all the tweets at 50 words so that the models could be less complex and easier to train. Tweets shorter than 50 words are padded with 0s either at the beginning or the end of the tweets, depending on the model used.

4 Embeddings

In order to have a distributed representation of the input words we used word embeddings. In particular we adopted `W2V` and `ELMo` in order to use both static and contextual word vectors. Since training these embeddings on an external dataset would be expensive and not the purpose or our work, we decided to use pretrained word embeddings both for `W2V` and `ELMo`.

4.1 Word2Vec

Regarding `W2V`, two different pretrained models has been used: `W2V` by Spacy and NLPL[5]. The former has been trained on different datasets: UD Italian ISDT v2.5, WikiNER, OSCAR crawler and Wikipedia (2020-03-01). It has a vocabulary of 500K words and an embedding size of 300. The latter has a vocabulary of approximately 2.5M words and 100 as embedding size and it has been trained trained on Italian CoNLL 17 corpus.

Since in sentimental analysis the polarity of a word is crucial and `W2V` is trained so that words with similar context have similar embeddings, for both models a fine tuning procedure has been applied. First, the words not already contained in the model vocabulary have been initialized with a random embedding. Then, the model has been finetuned using the working set (Section 5.1) for 30 epochs.

4.2 ELMo

Also in this case we used a pretrained model i.e. `ELMoForManyLangs`[6]. The architecture reflects the one proposed in the original paper[7], using a character CNN to represent the input words and using BiLSTM in the hidden layers. The model has been trained using a set of 20-million-words data randomly sampled from wikidump and common crawl.

Differently from what was done with `W2V`, `ELMo` embeddings have not been finetuned due to its computational cost and since it would have introduced high technical complexity.

5 Models

All the models tested on the task are presented in this section. For each model, we first present the architecture and techniques used and then show and discuss the experiment results. All the experiments have been ran on `Google Colab` using `GPU`s to speed up the trainings.
Section 5.1 describe the general model selection setup for all the models, while the following sections are dedicated to the single models.

5.1 Model selection

The model selection has been performed using k-fold cross validation technique with k equal to 5. For all the models, the same folds were used in order to make the comparison among models fair.
The score function that has been used is the F1-score since it is the one used for the task evaluation.

5.1.1 Early stopping

In order to regularize the model the early stopping technique has been used for all the experiments. The maximum number epoch has been set as 1000 while the patience has been set to 20. As for the model selection the metric used is the F1-score. The subset used to evaluate when to stop the training is the validation set itself, i.e. the fold not used in the training.

5.2 Dealing with unbalanced classes

Since the dataset is unbalanced towards `Non-Misogynous` tweets, two approaches were combined: weighted loss function and stratified mini-batch.

5.2.1 Weighted loss function

An approach widely used to deal with unbalanced classes is the usage of a weighted loss function, namely the introduction of weights inside the loss function according to the sample's class frequency in the dataset. Since the score function, i.e. the F1-score, is not differentiable it can't be used as loss function. Therefore, the loss function to use has been decided with an empirical analysis evaluating the result of the usage of the different functions. Those taken into consideration are: `Differentiable F1`, `Weighted Binary Cross-entropy`, `Dice`, `Dice + WBC`. The result of the described analysis are reported in appendix A.
The choice of the loss function has been made mainly based on two factor: the degree of similarity in the behaviour with respect to the score function and the learning speed and limitations. The best loss function turned out to be the `Dice` and, therefore, it has been used for all the tested models.

5.2.2 Stratified mini-batch

Since during the training the mini-batch approach has been used, the problem of class unbalance inside the mini-batch arises. This can produce not very robust sub-gradients. To overcome this issue the mini-batches have been produced using stratified sampling. Therefore, the stochastic algorithm for generating the mini-batches has been modified to maintain the classes proportion of the initial dataset.

5.3 Baseline

In order to compare and understand the degree of improvement of the subsequent models, a simple baseline model has been trained and tested. Figure 12 shows the baseline model. It consists of a neural network having an embeddings layer which maps words, as vocabulary indices, into a dense embedding array. The embeddings are then fed to a dense hidden layer followed by a 2 sigmoid units output layer.
Moreover, the same architecture has been changed using pre-trained embeddings, namely `Word2Vec` and `ELMo`.

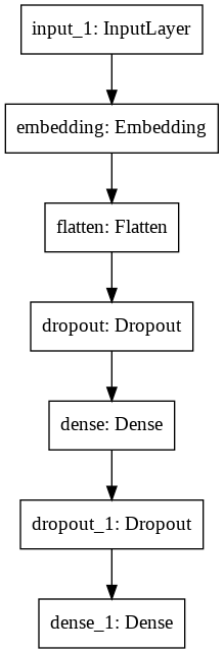

Figure 12: Baseline model

5.3.1 Experiments

Table 2 shows the hyperparameters used for the baseline model. In all the cases, the embeddings were fine-tuned together with the training of the model. `l2 regulariz.`, instead, specifies the regularization used in both layers. The other parameters follow

Parameter	Values
embedding	nlpl, ELMo
hidden size	32, 64, 128
l2 regulariz.	0.00001, 0.0001
learning rate	0.001, 0.0001

Table 2: Baseline hyperparameters

embeddings	emb.size	dropout	hidden s.	l2	lr	score
NLPL	100	0.5	32	0.0001	0.001	0.76665
ELMo	1024	0.5	128	0.00001	0.001	0.77859

Table 3: Baseline best models

straightforward from the model figure.
Table 3 shows the best model for each embedding tested. The random embeddings were used only in the preliminary experiments and therefore not reported. Among the two pretrained embeddings, ELMo performs slightly better, thanks to the contextual information contained in the embeddings. This shows the importance of contextual information for this task, which is, indeed, strictly related to sentiment analysis.

5.4 Convolutional models

We extensively tested convolutional models on the task, taking inspiration from the work in [8]. There, the authors present different models, all based on the application of the convolution over the vector representation of the words, in order to obtain a good feature set that could be used to solve the task at hand. We used the same idea, sometimes adding our contribution to improve the models.
Figure 13 shows the general architecture of the models. First, the word embeddings composing the sentence pass through a convolutional layer. Each filter in the convolution is applied on m subsequent word vectors, thus making it a $\mathbb{R}^{m \times k}$ matrix. The filter is applied on the whole sequence, resulting in a $\mathbb{R}^{n-m+1}$ feature map, with n equal to the length of the sequence, on which a max-over-time pooling operator is applied, returning a single feature. Of course, the layer uses multiple filters and multiple filter sizes to capture different features. All these are then concatenated in a single vector which is fed to a simple feed-forward neural network, with two sigmoid output unuts, one for each target.
As also [8] shows, different variants of this architecture can be used, which differ from what the convolutional layer gets as input and how many independent convolutional layers are used. In particular, we experimented with three versions of the model:

- **Single Channel**: Corresponds to the straightforward implementation of the model above. Here the choice can be made on whether training the embeddings or leaving them unchanged. In the experiments, we decided to train also the embeddings,

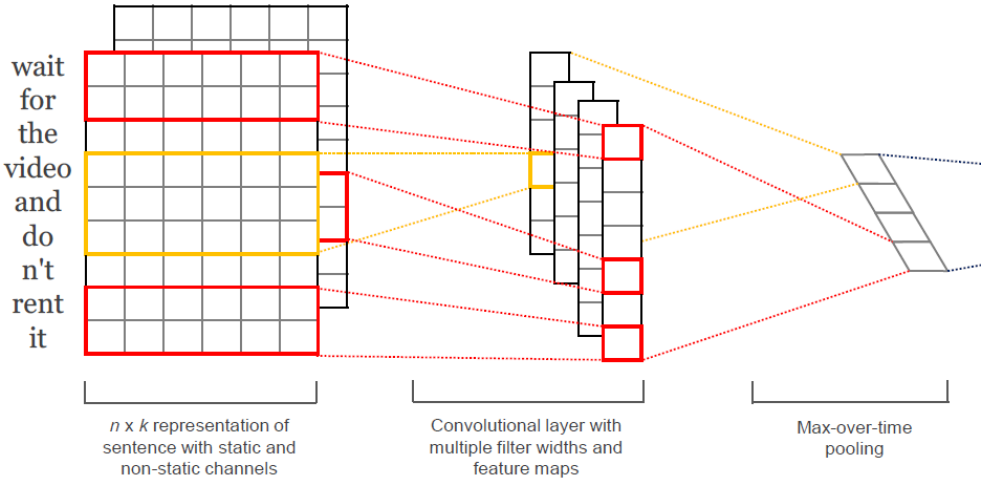

Figure 13: General Models Architecture

since many of them are semantic representation of the words, while our task also involves sentiment analysis.

- **Multi Channel**: Instead of either training the embeddings or leaving them unchanged, we mixed the contribution of both approaches, using two sets of embeddings: one left unchanged (namely, the *static* one) and one which is trained. Both embeddings pass through the same convolutional layer, thus the same filters are applied on both of them, while the pooling operator is applied on the two sets independently. In this way, two sets of features are produced: one that takes into account the simple semantic information while another that also contains sentimental information. We used the same set of filters on both the embeddings in order to obtain more general features, while at the same time, avoiding them to overfit for our specific task.

- **Contextual CNN**: Since most embeddings associate a vector to a word whatever context that word appears in, the context of a word is considered only in the convolutional layer. In order to consider such context also in the word representation itself, we used the same approach of `Multi Channel CNN` but using, as *static* information, `ELMo` embeddings (Section 4.2). In this case, since the two sets of embeddings could have different sizes, we used two distinct sets of filters, one for each embedding, while the MaxPooling operator is still applied so to have one output for each filter.

5.4.1 Experiments

The experiments with all the models followed the same setup described in Section 5.1. Moreover, for all the convolutional layers, a constraint on the norm of the filter matrix

Parameter	Values
kernel size	3-5, 3-5-7, 3-5-7-9
filters	25, 50, 75
dropout ratio	0.3, 0.5
l2 regulariz.	0.0005, 0.0001, 0.00005
learning rate	0.0005, 0.0001, 0.00005

Table 4: Single Channel CNN hyperparameters

have been used, with a fixed value of 3. Appendix B contains the images of the Tensorflow models for each architecture.

Table 4 shows the hyperparameters values for the `Single Channel CNN` grid search. `kernel size` specifies the list of filter sizes used in the convolutional layer. For each size, a number of different filters are trained, specified by `filters`, resulting in a total of `filters` $\times$ $\#sizes$ filters. A dropout layer has been used after the concatenation of all the pooling outputs, controlled by the `dropout ratio` parameter. For the feed-forward network, a single layer projecting the concatenation into two output units has been used, with `l2 regulariz.` being the regularization ratio used for the layer.
As embeddings, all three `NLPL`, `Spacy` and `ELMo` have been tested, using the same set of hyperparameters. In the first two cases, also the embeddings have been trained, while `ELMo` embeddings have been left unchanged, due to the incompatibility of ELMo model, implemented using PyTorch, and our models, implemented using Tensorflow.
Table 5 shows the three best models for each embeddings set. `NLPL` embeddings perform much better than the other two for the same hyperparameters. All the best configurations use 3 and 5 as filter sizes, meaning that most of the information is contained in a small neighbourhood of the word. However, this could also be due to the length of the tweets, which in the majority of the cases are too short to really exploit bigger contexts. The dropout ratio is also quite high (0.5) in most of the cases, meaning that

Model	kernel s.	filters	dropout	l2	lr	score
NLPL	3-5	50	0.5	0.00005	0.0005	0.83486
	3-5	50	0.5	0.0001	0.0001	0.83450
	3-5	25	0.3	0.0001	0.0005	0.83428
Spacy	3-5	25	0.5	0.0005	0.0005	0.81744
	3-5	25	0.5	0.0001	0.0005	0.81706
	3-5	25	0.5	0.00005	0.0005	0.81679
ELMo	3-5	75	0.5	0.0005	0.0005	0.81777
	3-5	25	0.5	0.00005	0.0001	0.81728
	3-5	25	0.5	0.0001	0.0005	0.81708

Table 5: Single Channel CNN best configurations

the model tends to overfit the data. However, reducing the number of filters resulted in worst performances than the ones reported here.

Parameter	Values
kernel size	3-5, 3-5-7, 3-5-7-9
filters	25, 50, 75
dropout ratio	0.5, 0.7
hidden size	50, 90, 125
l2 regulariz.	0.0001, 0.00001
learning rate	0.0005, 0.0001, 0.00005

Table 6: Multi Channel CNN and Contextual CNN hyperparameters

For what concerns `Multi Channel CNN`, Table 6 shows the hyperparameters values tested. These are the same used also for the `Contextual CNN`, as discussed later. The parameters are the same as before, but with adjusted values. However, given the bigger size of the concatenation of the pooling outputs, we decided to insert an hidden layer in the feed-forward network, in order to avoid a projection in a much smaller space, whose size is controlled by the parameter `hidden size`. As a consequence, the dropout ratio was increased. Again, lowering both the filters/hidden size and the dropout ratio resulted in worst performances than the ones reported here. As embeddings, since `ELMo` ones could not be fine-tuned, we tested only `NLPL` and `Spacy` embeddings.
Table 7 shows the best three models for the two embeddings. Again, `NLPL` embeddings perform much better than `Spacy`, using smaller filters. On the other side the number of distinct filters is usually lower for `Spacy` resulting in a smaller number of trainable parameters. However, there is not so much difference between the best model in this case and the best `Single Channel` model, meaning that all in all, the initial model was already powerful and regularized enough for the task.

Finally, Table 8 shows the results for `Contextual CNN`. The hyperparameters tested are the same used for `Multi Channel CNN`, in Table 6. In this case, only `NLPL` embeddings have been tested since they always performed better than the others in the previous experiments. Despite performing better than `Multi Channel CNN` with `Spacy`,

Model	kernel s.	filters	dropout	hidden s.	l2	lr	score
NLPL	3-5	75	0.5	90	0.0001	0.0001	0.83581
	3-5	75	0.5	125	0.0001	0.0005	0.83507
	3-5-7	25	0.5	125	0.0001	0.0005	0.83505
Spacy	3-5-7	25	0.5	90	0.0001	0.0005	0.81839
	3-5-7-9	25	0.5	90	0.00001	0.0001	0.81766
	3-5	50	0.5	125	0.00001	0.0005	0.81407

Table 7: Multi Channel CNN best configurations

it still can't beat the other models when NLPL embeddings are used.

Model	kernel s.	filters	dropout	hidden s.	l2	lr	score
NLPL	3-5	25	0.7	50	0.0001	0.0005	0.82951
	3-5	25	0.7	50	0.00001	0.0005	0.82928
	3-5	50	0.7	90	0.00001	0.0005	0.82849

Table 8: Contextual CNN best configurations

Figures 14 and 15 show the confusion matrices for the best CNN model, namely, Multi Channel CNN with NLPL embeddings. The results on the training set shows a very high accuracy for the Non-Misogynous and Aggressive classes, while many errors are made for Misogynous tweets. In particular, these are often confused with Aggressive tweets meaning that the model is not really able to distinguish when a tweet includes also aggressive behaviour, while it performs very well in identifying the misogyny.
On the validation set the results are similar, with an obvious worsening of the general accuracy. In this case, however, more than half of the Misogynous tweets are incorrectly classified as Aggressive, while another small percentage is classified as Non-Misogynous.

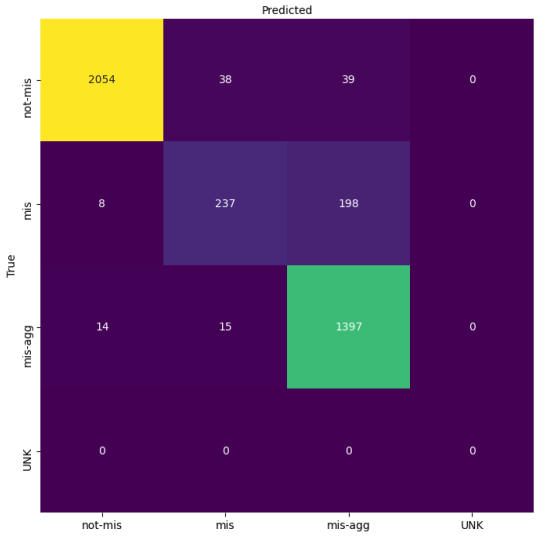

Figure 14: Confusion matrix for the training set

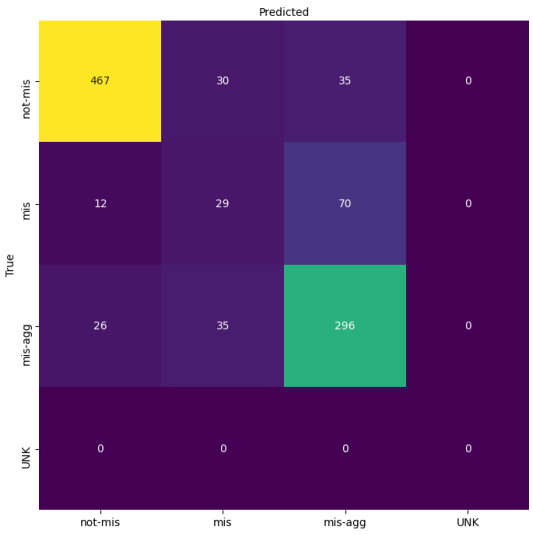

Figure 15: Confusion matrix for the validation set

5.5 Recurrent models

A further analysis has been performed using recurrent models, namely Recurrent Neural Networks. The structure of the model is summarized in Figure 16.

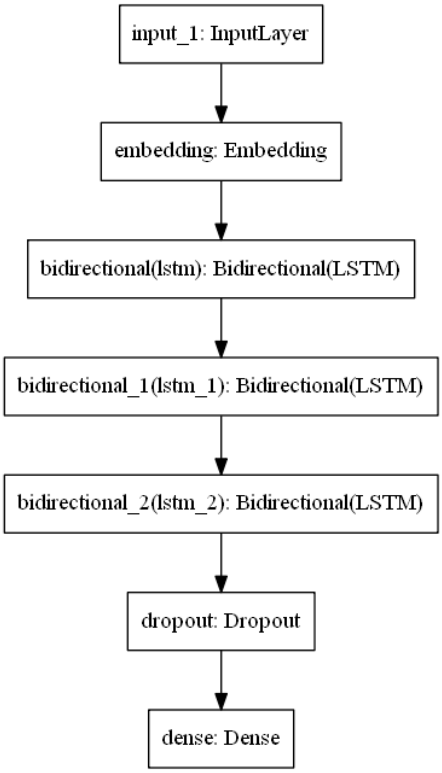

Figure 16: Recurrent model

5.5.1 Model structure

The model is a simple recurrent neural network model which takes as input the indexes of the words in the dictionary. The input passes through an embedding layer which produces a dense vector that feeds the recurrent layer. The architecture supports any type of recurrent layer and embedding layer. At the end there is a dropout layer that connects the last recurrent layer to the output layer, which is a dense layer with two sigmoid units.

5.5.2 Experiments

As for the other models, several preliminary experiments has been performed. Those showed very poor performances, together with a very slow and expensive training. Several hyperparameters, both structural (number of units, type of units etc..) and training ones (learning rate, dropout, etc..) have been tested. The best score obtained on the validation set (as average across folds) is 0.77005 using the parameters shown in table 9. It is possible to argue why the recurrent models perform poorly using some characteristics of the dataset and some theoretical characteristics of the recurrent neural networks. Since the RNN' s predictions highly depends on the internal state, i.e. contexts of the

Model	Number of units	type of units	dropout	lr	score
RNN	{64-32-16}	LSTM	0.3	0,0005	0.77005

Table 9: RNN best configuration

recurrent units it is useful to analyze this aspect.
The context for a recurrent unit is initialized to a zero vector and then, as the model sees new elements of the sequence, the correspondent feature becomes more and more meaningful. However, the dataset is a set of short sentences (due to tweets nature) so the context does not see enough data resulting in an inaccurate prediction.

5.6 Transformer

Considering the complexity of the task, we decided to test also some state-of-the-art model in NLP. We opted for BERT which is one of the best models for many NLP tasks. Since the full training of a BERT model takes a lot of computational resources and time, we decided to use a pretrained version. Many Italian BERT models are available online, but we focused on two specific versions: AlBERTo[4] and dbmz.
The first one, AlBERTo, is a BERT model for the Italian language focused on the language used in Twitter. It has been trained on the EVALITA 2016 task SENTIPOLC and, differently from the original BERT, it has been trained only on the Masked LM task, leaving aside the Next Sentence Prediction task.
The second one, `dbmz`, is an independently developed version of BERT, trained on a Wikipedia dump and various texts from the OPUS corpora collection.
In both the cases, the prediction was computed based on the final hidden state corresponding to the `[CLS]` token, namely C using a single 2 sigmoid unit output layer.

5.6.1 Experiments

In the preliminary experiments `dbmz` resulted immediately less performing than many other tested models, while AlBERTo performed very well. The reason could be because AlBERTo is trained on Twitter data and hence is more capable of dealing with the specific language used in this setting, while dbmz, trained on a different kind of corpus, is not able to deal with Twitter language. Therefore, in the following, we focus on only AlBERTo and its results.
Since the training of the model has a high computational cost, we tried a restricted subset of hyperparameters thanks also to the preliminary experiments. The hyperparameters

Parameter	Values
l2 regulariz.	0.0001
learning rate	0.00005, 0.00001, 0.000005

Table 10: AlBERTo hyperparameters

Model	l2 regulariz.	learning rate	score
AlBERTo	0.0001	0.00001	0.8450
	0.0001	0.000005	0.8378
	0.0001	0.00005	0.8112

Table 11: AlBERTo result

are reported in table 10.
Table 11 shows the result for AlBERTo model. The best configuration has as score 0.84503, which is the best obtain so far.

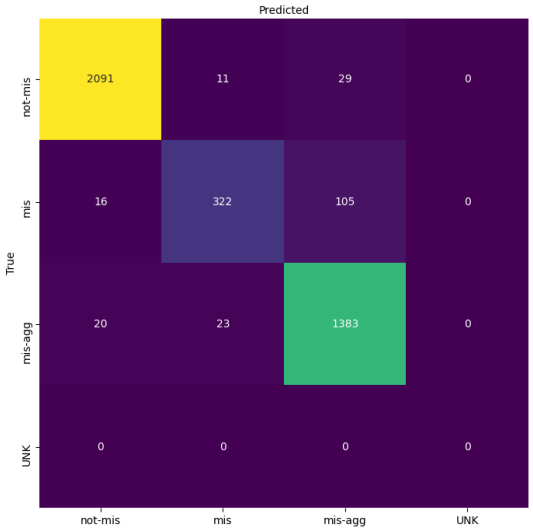

Figure 17: Confusion matrix for the training set

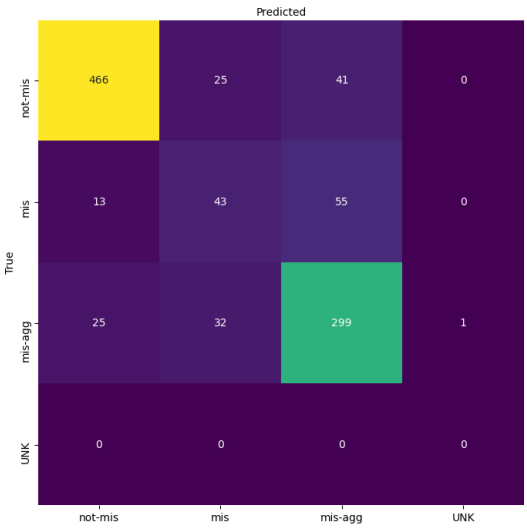

Figure 18: Confusion matrix for the validation set

Figures 17 and 18 show the confusion matrices for AlBERTo. Again, most of the errors are due to `Misogynous` tweets classified as `Aggressive`, both in the case of the training set and the validation set. The performances are, however, slightly better than those of CNN, which is supported also by the higher score obtained in this case.
Interestingly, on the validation set, one `Aggressive` example is labeled as `Unknown`, meaning that it obtained a 1 in the Aggressiveness column but a 0 in the Misogyny column, which is not allowed in this dataset. This, however, is not surprising considering that the outputs are trained independently and, therefore, nothing stops such an output to be returned.

5.7 Mixture of experts

In order to improve the performance using different features extracted by the best models seen so far a new model has been created, based on the idea of the mixture of expert

technique. The model just mentioned will be refers from now on as `MoE`. The structure of the model is summarized in the Figure 19.

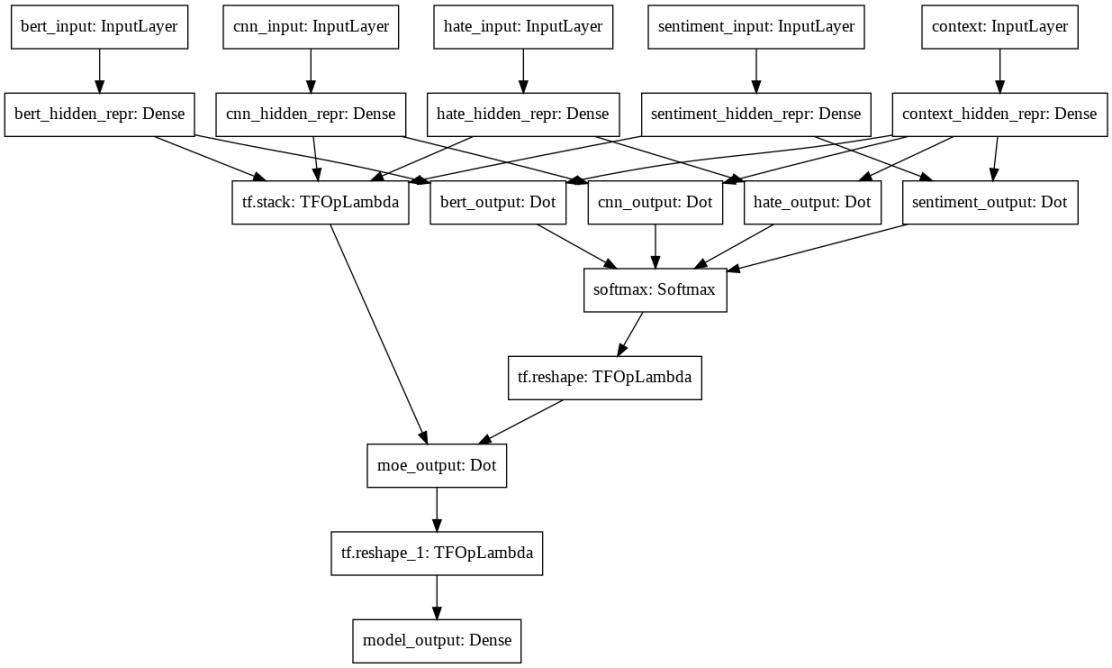

Figure 19: Mixture of experts model

5.7.1 The MoE model

This section contains a brief recall on how the MoE technique works with neural network models. In particular the focus is on feature-based attention model. The idea behind is simple: different experts can learn different feature vectors, for instance different experts can learn different grammatical rules or focus on extracting different semantic characteristics. In order to combine the features extracted by the experts the attention mechanism is used which, based on a context information, decides how much importance to give to those features. In NLP for instance the context can be a feature representation of the sentence.
From now on we will refer to the features extracted by the experts as inputs and we will treat the context separately for the sake of clarity although they are all inputs of the MoE model.

5.7.2 Input and context

The input of the model can be partioned in two sets:

- **Task experts**:

  - `cnn_input`: feature vector extracted by the best CNN model
  - `bert_input`: feature vector extracted by the best BERT model

- **Non-task experts**:
  - `hate_input`: feature vector extracted by an expert used to dected hatespeech in Italian language. It has been trained by finetuning multilingual BERT ([9])
  - `sentiment_input`: feature vector extracted by an expert used for sentiment analysis on Italian sentences. It has been trained on bert-base-italian-cased and finetuned on a dataset of Italian tweets ([10])

The partitioning highlights an advantage of the MoE's structure that allows the usage of both models trained on the task and those trained on different datasets.
Note that the task experts have been chosen based on an analysis of the best models obtained according to the validation schema, while the non-task experts have been chosen based on the similarity to the task or information relevant to it.
The context used for an input instance is a feature vector obtained stacking all the `ELMo` embedding for each word contained in the sentence.

5.7.3 Projection in homogeneous regions

Since the different experts produce feature vectors with different sizes and MoE requires feature vectors with the same size, a projection in the same vectorial space is required. The projection has been performed with a neural network for each input with the same output size, i.e. the size of the feature vector that are used as input of MoE.

5.7.4 Output

The output of the attention module has been used as input for the output layer, which is a dense layer with two sigmoid units.

5.7.5 Experiments

The focus of the experiments phase for MoE was on three hyperparameters: the `hidden representation` that represents the size of the space in which the different feature vectors are projected into (Section 5.7.3), `l2 regularization` and `learning rate` which control the learning phase. The hyperparameters are reported in Table 12.

Parameter	Values
`hidden representation`	100, 200, 300
`l2 regulariz.`	0.0005, 0.0001, 0.00005
`learning rate`	0.0005, 0.0001, 0.00005

Table 12: MoE hyperparameters

Table 13 shows the result for the MoE model. The best configuration achieves a score of 0.84607, which is the best obtained.

Model	`hidden representation`	`l2 regulariz.`	`learning rate`	`score`
`MoE`	200	0.0001	0.0005	0.84607
	200	0.0005	0.00005	0.84567
	200	0.0005	0.0005	0.84490

Table 13: MoE result

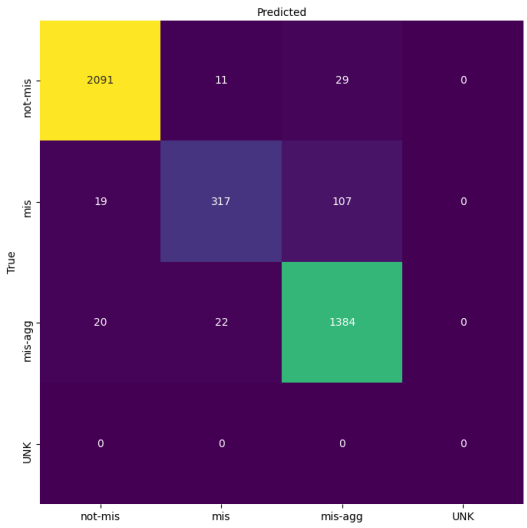

Figure 20: Confusion matrix for the training set

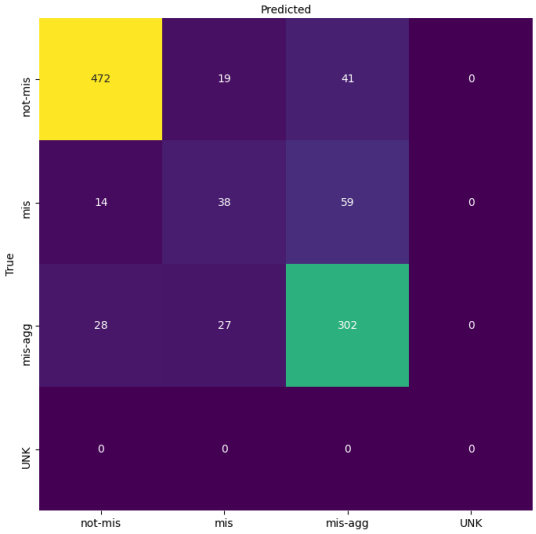

Figure 21: Confusion matrix for the validation set

Figures 20 and 21 are the confusion matrices computed on the training and on the validation set respectively. The figures show the same behaviours of AlBERTo model although the performances are slightly better.

6 Final results

This section reports the results of the model selection (Section 6.1) and describes the model assessment (Section 6.2).

6.1 Model selection

Table 14 shows the best results for each model. Among them, the best one according to the validation schema is MoE hence it was the input of the model assessment.

Algorithm	**VL avg**
`Baseline`	0.8142
`CNN-single`	0.8349
`CNN-multi`	0.8358
`CNN-contextual`	0.8295
`RNN`	0.7700
`BERT`	0.8450
`MoE`	**0.8460**

Table 14: Model selection result

6.2 Model assessment

The final model has been retrained on all the training data keeping a portion (15%) for the validation used for early stopping. Results are reported in Table 15.

VL score	0.859	**TS score**	0.718

Table 15: Final model result

Finally Table 16 shows a comparison of the obtained results with the best results obtained in the competition. [u] denotes the teams that have used additional data to train the model, producing two separate rankings. Since we didn't use additional data, we are part of the second ranking in which we finished second.

Algorithm	TS score
jigsaw [u]	0.740
fabsam	0.738
YNU_OXZ [u]	0.731
Our final model	0.718
NoPlaceForHateSpeech	0.717
AMI_the_winner	0.687
MDD [u]	0.684
PoliTeam	0.683
AriannaMuti	0.634

Table 16: Competition result

7 Conclusion

We addressed the task of Automatic Misogyny Identification testing several models and improving them with iterative and incremental enhancements to their structure and learning phase.
Starting from the baseline, we implemented different models, exploiting different approaches and trying to understand how they could be improved. Some models immediately resulted not suitable for the task, while others gave more interesting results.
We also tested some state-of-the-art model on the task, to see whether these models were actually able to outperform other simpler but more specific model. In our case, the performances did not substantially change deploying these models.
Finally, we used an ensemble approach to see whether the performances could further improve by exploiting the best of each model. However, even in this case, the approach did not lead to a substantial improvement.

Considering that the performances did not improve a lot deploying more complex models, some improvements could be obtained enhancing the preprocessing phase. Indeed, it has been defined ad hoc for the dataset, therefore, implemented from scratch, whereby using a more general and standard preprocessing, e.g. using established libraries, could produce better results, especially on pretrained models which may have been trained using data proprocessed with standard techniques. A good compromise could be a prepossessing procedure that merges the goods of our custom preprocessing phase and the standard one.

A final note must be done on the quality of the datasets. In particular there is a remarkable difference in the style of the training and the test instances. Indeed, the former tend to be simpler and less articulated, especially in the case of `Misogynous` and `Aggressive` tweets, while the latter are generally well written and their contexts are meaningful no matter the label of the example. This reflects the generally poor results

obtained in the competition by all the groups.

References

[1] Lucas Dixon, John Li, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Measuring and mitigating unintended bias in text classification. In Jason Furman, Gary E. Marchant, Huw Price, and Francesca Rossi, editors, *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2018, New Orleans, LA, USA, February 02-03, 2018*, pages 67–73. ACM, 2018.

[2] Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Nuanced metrics for measuring unintended bias with real data for text classification. In Sihem Amer-Yahia, Mohammad Mahdian, Ashish Goel, Geert-Jan Houben, Kristina Lerman, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *Companion of The 2019 World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 491–500. ACM, 2019.

[3] Debora Nozza, Claudia Volpetti, and Elisabetta Fersini. Unintended bias in misogyny detection. In Payam M. Barnaghi, Georg Gottlob, Yannis Manolopoulos, Theodoros Tzouramanis, and Athena Vakali, editors, *2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, Thessaloniki, Greece, October 14-17, 2019*, pages 149–155. ACM, 2019.

[4] Marco Polignano, Pierpaolo Basile, Marco de Gemmis, Giovanni Semeraro, and Valerio Basile. Alberto: Italian BERT language understanding model for NLP challenging tasks based on tweets. In Raffaella Bernardi, Roberto Navigli, and Giovanni Semeraro, editors, *Proceedings of the Sixth Italian Conference on Computational Linguistics, Bari, Italy, November 13-15, 2019*, volume 2481 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[5] Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In Jörg Tiedemann and Nina Tahmasebi, editors, *Proceedings of the 21st Nordic Conference on Computational Linguistics, NODALIDA 2017, Gothenburg, Sweden, May 22-24, 2017*, volume 131 of *Linköping Electronic Conference Proceedings*, pages 271–276. Linköping University Electronic Press / Association for Computational Linguistics, 2017.

[6] Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In Daniel Zeman and Jan Hajic, editors, *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, October 31 - November 1, 2018*, pages 55–64. Association for Computational Linguistics, 2018.

[7] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the*

2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), pages 2227–2237. Association for Computational Linguistics, 2018.

[8] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751. ACL, 2014.

[9] Sai Saketh Aluru, Binny Mathew, Punyajoy Saha, and Animesh Mukherjee. Deep learning models for multilingual hate speech detection. *CoRR*, abs/2004.06465, 2020.

[10] Neuraly S.R.L.S. + neuraly - italian bert sentiment model. *huggingface*.

A Loss function analysis's plots

The results below were generated using the baseline model, since the choice of the loss function was prior to advanced model analysis, although the choice of dice has been continuously under critical analysis even with respect to the other models.
All the plots below report the ratio between two time-steps using a specific loss function i.e.:

$$\frac{L(i) - L(i-1)}{L(i-1)}$$

where L(i) is the loss computed at time step i.

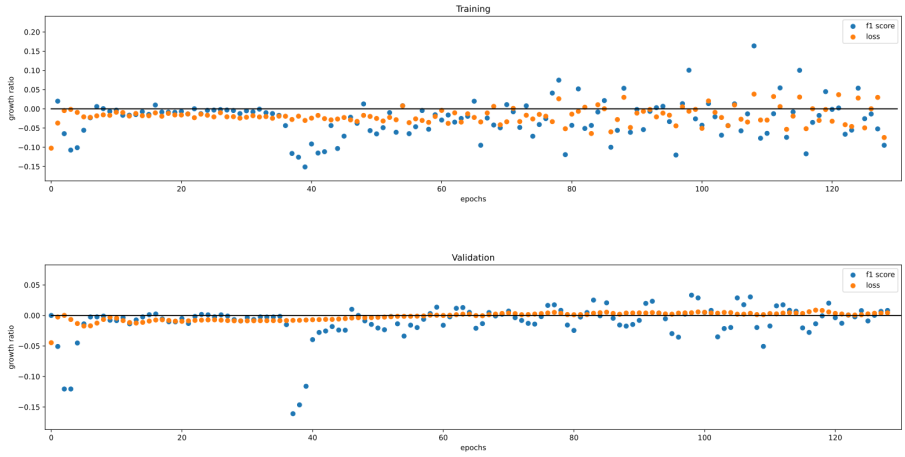

Figure 22: Result obtained with `Weighted Binary Cross-entropy`

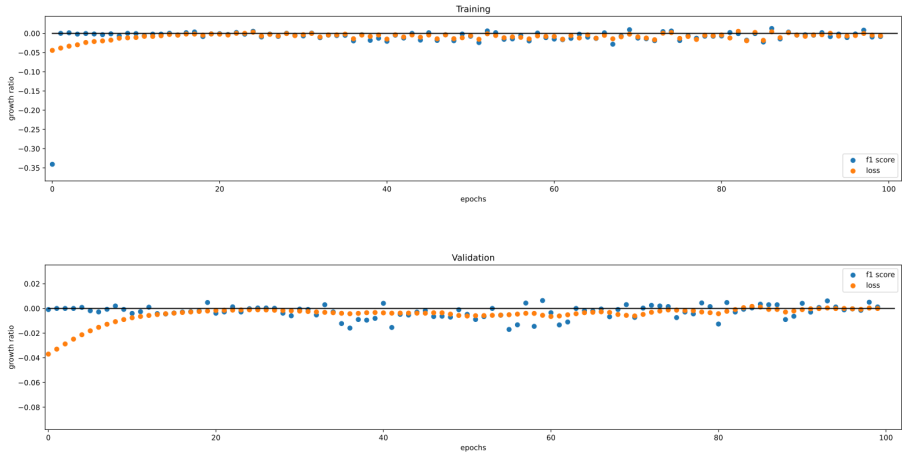

Figure 23: Result obtained with `Differentiable F1`

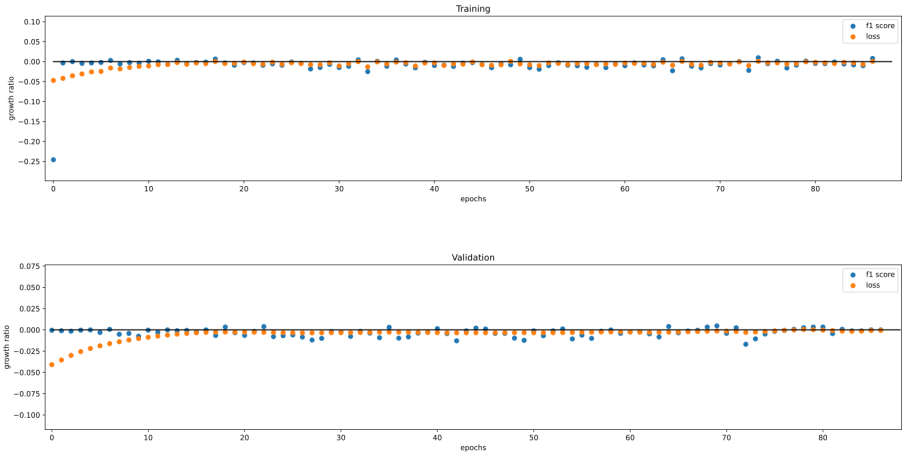

Figure 24: Result obtained with `Dice`

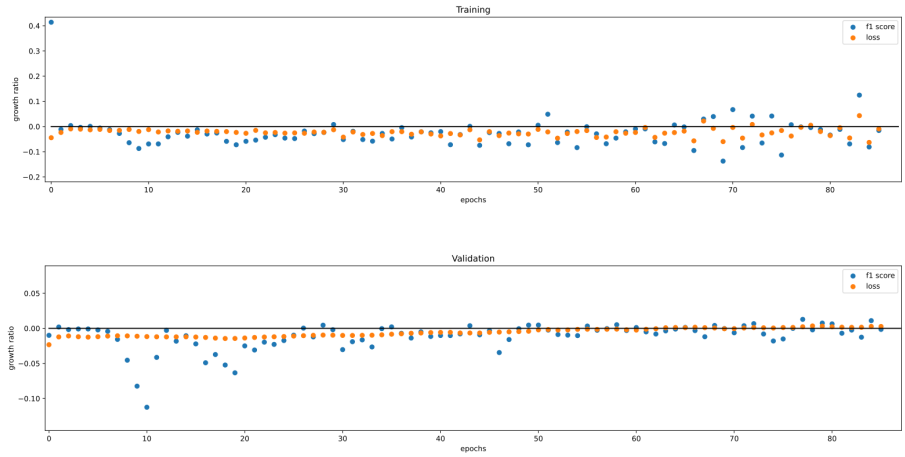

Figure 25: Result obtained with `Dice + WBC`

B Convolutional architectures

In this section are reported all the figures related to the convolutional models 5.4.

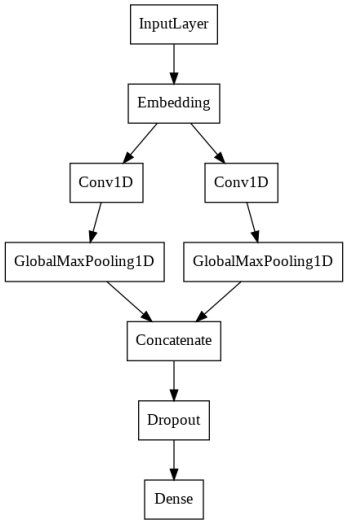

Figure 26: Single channel CNN

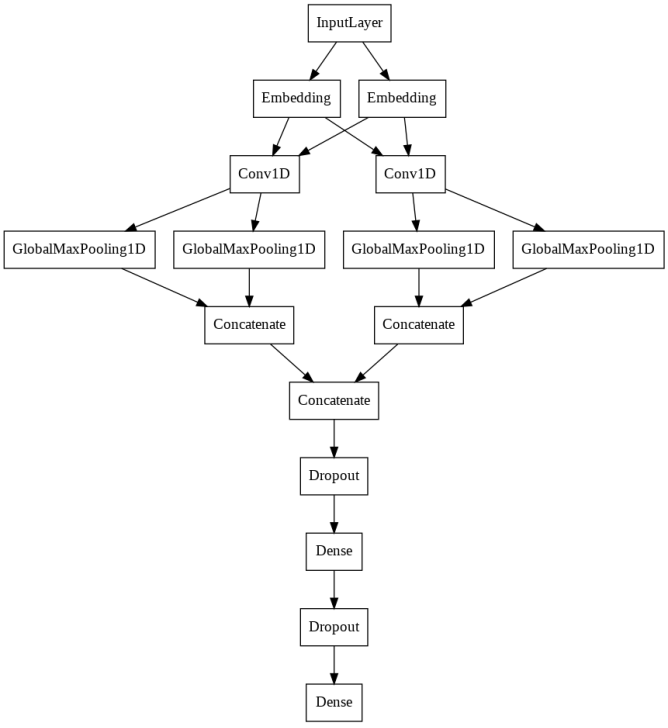

Figure 27: Multi channel CNN

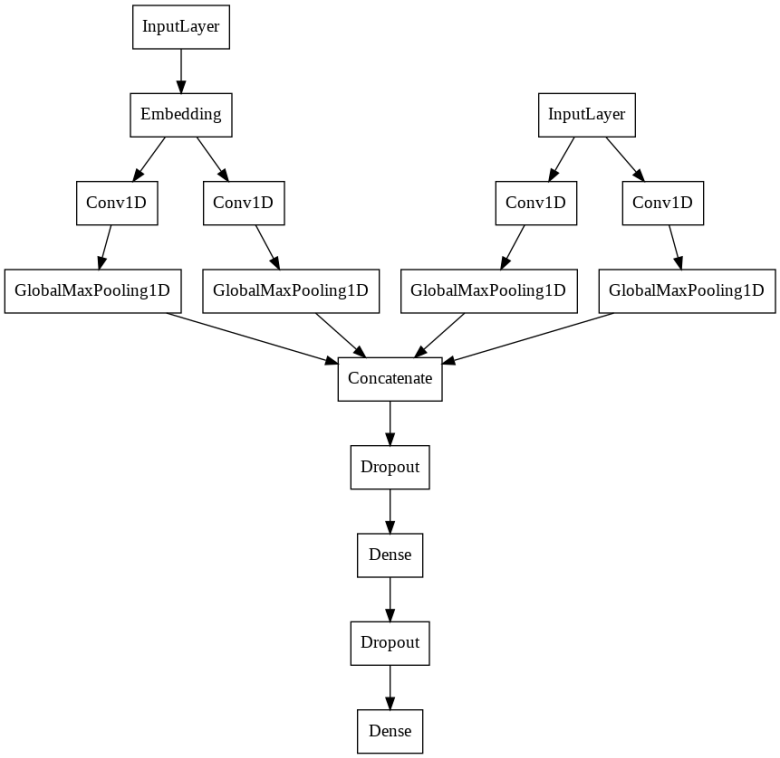

Figure 28: Contextual CNN