HaSpeeDe: Exploring the Performance of Hate Speech Detection Models on Different Datasets and Languages

NLP Course Project & Project Work

Martina Ianaro, Davide Perozzi, Fabian Vincenzi
Master's Degree in Artificial Intelligence, University of Bologna
{ martina.ianaro, davide.perozzi, fabian.vincenzi }@studio.unibo.it

Abstract

This study presents the results of a binary classification task on hate speech detection for different datasets, including tweets, news, and Nominal Utterances dataset. The performance of the models was evaluated based on precision, recall, and F1 score. The results showed that the models performed better on datasets with a similar positive/negative label ratio to the train set.
One limitation of the study is the limited size of the German datasets, which may impact the results. Future work can involve expanding the size of the datasets to include more diverse samples of hate speech and improove the labels balance. Additionally, the pre-processing of the tweet text, including the substitution of emojis with their word equivalents and the breaking of hashtag words, can improved models performance. Overall, this study provides insights into the performance of hate speech detection models and highlights potential areas for improvement in future research.

1 Introduction

Hate speech detection is a crucial task that can be analyzed in its three main phenomena, as summarized in the tasks solved in this paper. The primary task of hate speech detection is a binary classification task, in which a given message can contain hate speech or not. The task B is a binary classification task in which stereotypes are searched for in the text, as stereotypes often express prejudiced attitudes and uncritical judgments, which are synonymous with hateful messages. Both tasks above contain a cross-domain evaluation over news datasets, which are not fine-tuned. The task C is a named entity recognition (NER) task that recognizes nominal utterances (NU) in hateful tweets, where NU is a non-verbal syntactic construction that has a significant meaning in the expression of hate.

In this paper, we aim to identify custom entities from text data using NER to solve the task A enriched with a multilingual extension on a hate speech dataset of tweets in Spanish and German languages. The identification of online hateful content, due to its potential to be hurtful, is a crucial mission in any field. Offensive and abusive language has received attention, and its identification has gained prominence. Detecting hate speech reduces its negative impact and influence on users. This binary task is solved on Twitter datasets in the Italian language containing specific keywords related to Italian minorities. NLP plays a central role in detecting hate speech concerning religion, rage, gender or sexual orientation discrimination. Hate speech poses a significant threat to communities by instilling hatred in young people against others or by instigating criminal activity or violence.

Social media users often employ abbreviations and ordinary words to express their hate intent implicitly, known as code words, to evade detection, which adds extra difficulties to detect hate speech. The leveraged domain-specific word embedding in Bidirectional LSTM-based deep models and the use of the transfer learning language model (BERT) on the hate speech problem as a binary classification task are two possible ways. Experiments on these two approaches demonstrate that domain-specific word embedding with the Bidirectional LSTM-based deep model achieved a 93 % F1-score, while BERT achieved up to 96 % F1-score on a combined balanced dataset from available hate speech datasets.

Taking the results above as starting point, we solve the task exploiting BERT model.
The notebook related to this report try to solve the challenge HaSpeeDe2.

Each task is composed by four sections :

- Pre-processing;
- Tokenization;

- Model & functions;
- Grid Search;
- Training and Test with best hyper-parameters.

The HaSpeeDe dataset is used for tasks A and B, which includes tweet texts targeting minority groups with columns for hate speech and stereotype labels. For task C, the training set is annotated in WebAnno TSV format with NU annotations in an IOB format. The multilingual extension involves repeating the steps with different datasets. For German, the dataset contains a corpus of annotated tweets about refugees in Germany, while the Spanish dataset is derived from SemEval19, focusing on hate speech against immigrants and women in Spanish tweets from Twitter. Observations are made about the model's performance when fine-tuned with a dataset of tweet text structure over the same format test set and news format test set. The NER model has some difficulty in detecting certain NU tokens compared to others. Additionally, the multilingual extension demonstrates better performance on the Spanish Twitter dataset compared to those in German and Italian.

2 Background

Analysing the literature around Hate Speech (HS) detection task, it becomes clear that transformer-based models such as BERT outperform all other competitive embedding models. In fact, a scientific paper titled Detection-HS reported the performance of a deep model trained with BERT and evaluated it using metrics like weighted precision, recall, AUC, and f1-score. The f1-score was reported for each class separately to provide a clear insight into the classifier performance on each class. Other sources confirm that, in terms of transformer embeddings, the use of BERT provides the best results and consistently shows superior performance over competing models that can be Glove embedding-based CNN or MLP or Bi-LSTM and TF-IDF embedding-based SVM or MLP (website DeepLearning-HS). Furthermore, previous comparisons between two possibilities useful to solve binary classification tasks have been made between a bidirectional LSTM and LR approaches. A Hate Speech Word2Vec (HSW2V) as features and Bidirectional LSTM based deep model as a classifier overcomes the Logistic Regression classification approach that usually takes quite a long time to train and does overfit respect to Neural Network which is fast, has high accuracy and low overfitting. In the context of Name Entity Recognition (NER), the literature suggests three approaches to solve the task:

- rule-based for commercial system;
- The feature-based Hidden Markov Model (HMM) or Conditional Random Field (CRF) ;
- the neural network approach, which used to be BiRNN + HMM or CRF before the popularity of transformer-based models like BERT or NER from HuggingFace.

Recent studies have demonstrated that transformer-based models such as BERT have achieved state-of-the-art results on NER tasks across different domains, including the biomedical and general domain. This is likely due to their ability to capture complex patterns and dependencies in text data. In conclusion, we have chosen to solve the NER task with a transformer-based model, specifically BERT, due to its superior performance and versatility across different domains. Other transformer-based models, such as RoBERTa and XLNet, have also demonstrated strong performance on NER tasks and may be worth considering in future work. It is important to note that while hate speech recognition has made significant progress in recent years, it has also brought about controversy over the explainability of classification. However, in literature, LIME has been used as an effective approach for providing useful explanations in the context of hate speech detection. For instance, the study titled Explaining Hate Speech Detection Models with Local Interpretable Model-Agnostic Explanations used LIME to explain the output of a hate speech classifier based on a convolutional neural network (CNN) model. The authors found that LIME provided valuable insights into the model's decision-making process and helped identify areas of bias in the data.

3 System description

The models used for the tasks are based on pre-trained BERT (Figure 1, Figure 2):

- Task A, B and C - AutoModel and BertTokenizer from pretrained('dbmdz/bert-base-italian-uncased');

- Spanish extension - AutoModel and AutoTokenizer from pretrained("dccuchile/distilbert-base-spanish-uncased");
- German extention - AutoModel and BertTokenizer from pretrained("dbmdz/bert-base-german-uncased");

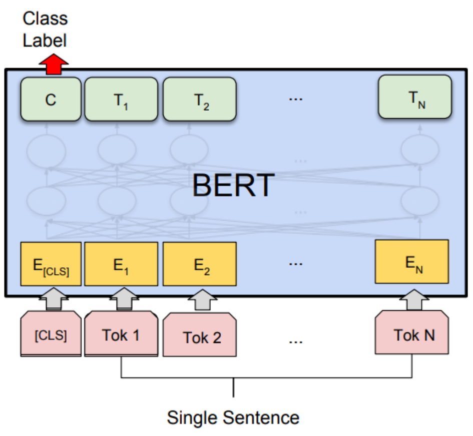

Figure 1: Bert architecture for binary classification

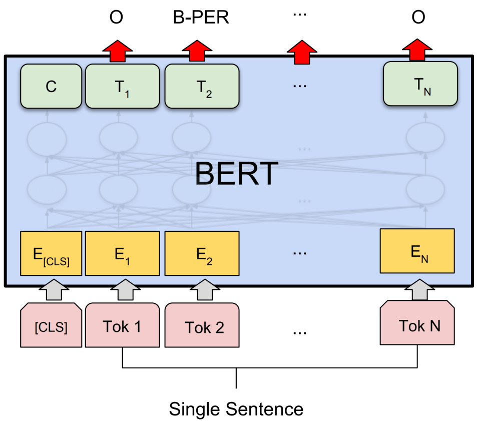

Figure 2: NER-BERT based architecture

The source data for the Italian BERT model consists of a recent Wikipedia dump and various texts from the OPUS corpora collection. The training corpus has a size of 13GB and 2,050,057,573 tokens. NLTK is used for sentence splitting. The uncased model is trained with an initial sequence length of 512 sub-words for around 2-3M steps. The German pre-trained model has as source data a dataset with a size of 16GB and 2,350,234,427 tokens. For sentence splitting have been used spacy. The pre-processing steps follow those used for training SciBERT. The model is trained with an initial sequence length of 512 sub-words and was performed for 1.5M steps. No specific informations have been provided by the authors of the Spanish pre-trained model.

Our models are built in PyTorch with dropout technique that randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution to avoid regularization and preventing the co-adaptation of neurons.

When inizialized, every model recieves a dropout value and the NER model also recieves the number of labels to predict; for each token, it has to return a BIO annotation between 3 possible labels, it is a multi-class classification task.

For each model the `forward` method specifies the model computation logic that transforms input tensors into output tensors across the layers of the neural network. This method is called automatically when calling the model with an input and returns the corresponding output.

In the binary classification model classes `forward` method, the pooled_output is computed from the model outputs, which is a single vector representation of the entire input sequence. This pooled_output is then passed through a dropout layer and a linear layer to produce the final output logits.

In the NER model class `forward` method, the sequence_output is computed from the BERT outputs, which is a sequence of vectors representing each token in the input sequence. This sequence_output is then passed through a dropout layer and a linear layer to produce the final output logits. Another difference is that the second takes an additional argument num_labels, which is the number of labels in the classification task, and uses this to determine the size of the final linear layer, while the first model has a fixed output size of 1.

Our training routine begin with the `train_model` function that uses `train_fn` and `val_fn` functions. Our `train_fn` function starts by setting the model to train mode and initializing the train loss to 0. Then it iterates over the batches in data_loader, zeroing the optimizer's gradients and passing the ids and mask tensors through the model to obtain the output logits. It then computes the loss using the criterion function and adds it to the train loss. After computing the loss, the code performs the backward pass to compute the gradients of the loss with respect to the model parameters. The optimizer then updates

the model parameters using these gradients. Finally, the function returns the average train loss across all batches.
The `val_fn` function begins by setting the model to evaluation mode using the `model.eval` method. It initializes the val_loss, predictions, and true_labels variables to 0 and empty lists respectively. Then, for each batch in the validation set, the function performs a forward pass through the neural network using the model(ids.to(device), mask.to(device)) statement. It computes the loss using the given criterion function and the predicted output and target labels. The function also calculates the validation loss by taking the average of all the batch losses. It then applies the sigmoid function to the output of the model and converts the values to 0 or 1 by thresholding at 0.5. The predicted values and true labels are appended to their respective lists. Finally, the function returns the average validation loss and the classification report computed by the classification_report function from the sklearn library, which provides a detailed evaluation of the model's performance on the validation set.
The NER model for task C has differences in the `val_fn` method because the predictions are the results of `torch.argmax` method applied on the model logits that returns the index of the most valuable element in a tensor.
In the `train_model` function we also use the early stopping to stop training once the model performance stops improving on a hold out validation dataset, the patience is based on the number of total ephocs (e.i. epochs*0.1). It avoids over-training because too many epochs can lead to over-fitting of the model over the training dataset, whereas too few may result in an underfit model. The early stopping helps to find the best balance between the generalization capacity of the model and its efficient learning capacity, checking the number of epochs that are passed without improvement on the validation set.
In Spanish and German extension, the models are structured equally to task A and B because the task kind is the same (binary classification) but with customized pre-trained model based on the language.
Following our pipeline after the pre-processing, tokenization and model definition, the grid search step is applied. Before training the model with best hyperparameters, we set the random seed. The `set_reproducibility` function sets the seed of the random number generators. The seed is a number that allows you to generate a sequence of random numbers. By setting the same seed, you will always get the same sequence of random numbers. `The set_reproducibility` function is used to set a start value for random number generation. When using data to train, validate, and test a machine learning model, it is important to maintain consistency and replicability of random number generation operations. This means that the model will always use the same training, validation, and test data when run at different times. This is important to ensure consistency of model results and to allow comparison of model performance under different conditions. Also, this helps prevent over-fitting problems from occurring. In Grid Search, we set our hyper-parameters grid specifying seed values ([42, 12321]), batch sizes ([32, 64]) and dropout rate ([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]). The learning rate, pos_weight and number of epochs are fixed. The pos_weight value have been defined as the ratio negative_labels/positive_labels in train set, so every language task has a different pos_weight based on specific dataset ratio. For the fixed values we choose some standard values.
In the last phase, Training and Test with best hyper-parameters, we run the training process a last time with the best hyper-parameters found out from the Grid Search and then evaluate the resulting final model with the `val_fn` function over the test set.
Our work builds upon existing research and heavily draws inspiration from various sources:

- architecture: Transformer-based model BERT;
- model architecture and training phase: Training with PyTorch;
- NER task: NER-Kaggle

4 Data

The HaSpeeDe Italian dataset, used for the challenge tasks, is publicly available (Manuela et al., 2020). The dataset contains tweets related to three main Italian minority topics: Immigrants, Muslim, and Rom communities, which are highly cited in tweet posts for their political resonance. These topics have become social problems that feed political

and public debates, and have highlighted the necessity for hate speech detection.
The dataset is split into a dev set, which is further divided into a train and validation set with an 80/20 percentage split, and a separate test set. The dev set consists of 27,348 tweets, while the test set contains 5,048 tweets and includes in-domain and out-of-domain data. The dev set's tab-separated values file provides four attributes per tweet, including the tweet ID, the tweet's Italian-language text, and two binary columns representing hate speech and stereotypes, respectively. The test set only contains values for the tweet ID and text.
Data analysis shows that the data distribution between the two classes of hate and non-hate speech tweets is imbalanced in the training and validation sets, with a majority of non-hate speech tweets (around 4,000) compared to 2,500 hate speech tweets. In contrast, the test set's two classes have similar cardinalities, with a slightly higher frequency of non-hate speech tweets (around 600).
In the pre-processing phase, the text column of the training dataframe serves as the X train dataframe, while the binary values of the "hs" column make up the y labels train dataframe. The X train dataframe is then split into train and validation sets with an 80/20 percentage split. Data analysis shows that there is a higher frequency of non-hate speech tweets in both the training and validation sets, with more than 3,000 non-hate speech tweets and 2,250 hate speech tweets in the training set, and 800 non-hate speech tweets and 550 hate speech tweets in the validation set.
Additionally, the challenge tasks include an extension with a cross-domain evaluation over a test set containing newspaper headlines. This evaluation aims to assess how well the models perform when trained on tweets and applied to a different textual genre. Data analysis of this test set shows that the data distribution between the two classes of hate and non-hate speech tweets is also imbalanced, with a majority of non-hate speech tweets (around 300) compared to 150 hate speech tweets.
In following table we consider also the characteristic of this dataset. Ratio positive/negative of task A:

Set	Train	Val	Test	News-Test
P	40.52 %	39.55 %	49.21%	36.07%
N	59.32 %	60.45 %	50.79%	63.93%

Only the test set is balanced.
Similar situation for Task B extracting the column "stereotype".

Set	Train	Val	Test	News-Test
P	44.54 %	44.23 %	45.09%	35.07%
N	55.46 %	55.77 %	54.91%	64.93%

After downloading and building the dataframe, the first step of our pipeline is the preprocessing. We decided to perform these manipulations to data: lower case, remove 'url' word, remove mentions by using regular expressions, remove non-alphanumeric characters by using regular expressions, remove duplicate whitespace and stopwords. Only lower case was performed for the NER task. To implement this preprocessing, we created a function called `preprocess_tweet` that takes a tweet as input and applies each of these steps sequentially. The function first converts the tweet to lowercase using the `lower` method. It then removes 'url' by replacing them with the empty string ", and removes mentions using regular expressions. Non-alphanumeric characters are removed using another regular expression, and duplicate whitespace and stopwords are removed by joining the words of the tweet after filtering out any stopwords.
After preprocessing we managed to tokenize the input text and codify the labels for the NER task: This is done by the function `tokenization` that uses the BertTokenizer from the transformers library to tokenize the input texts. If tokenizer is not provided, the function uses the default pre-trained Italian BertTokenizer provided by dbmdz/bert-base-italian-uncased.
Then, for each text and its corresponding label, the function encodes the text using the tokenizer, pads the encoded sequence to the maximum length using the pad_to_max_length=True and padding='max_length' arguments, and returns an attention mask for the encoded sequence.
The function also converts the label to a numerical representation using a dictionary called label_map. The label_map maps each label in y to a numerical value. The function then adds a PAD label at the beginning and end of the label list, pads the list to the maximum length, and appends the padded list to a list of label_ids.
Finally, the function converts the input_ids, attention_masks, and label_ids lists to PyTorch tensors and returns a DataLoader object containing the en-

coded input sequence, attention masks, and label IDs. The DataLoader object is used to load the data in batches during training or testing.

The tokenization for tasks A, B and Spanish has max length 256 due to the fact that all the instances in our sets are shorter than 256 while for task C and German the tweets have length lower than that 140. The max length of the German tokenizer is kept as 256 to make the results more comparable with task A, B and Spanish.

The Task C dataset is originally annotated with tweet ID, token number, token, and IOB annotation, where labels are denoted as B, I, and O. To reorganize the dataset, the training data was separated into two columns, one for tweet ID and one for token number, and the IOB annotation was reduced to its first character, "B", "I", or "O". Upon observing the training and test sets, it was found that there were a obvious majority of "O" labeled tokens in comparison to the 20,000 "I" tokens and a very low quantity of "B" tokens.

The pre-processing step for this dataset is more complex due to the reorganization of labels into four values: 'O':0, 'B':1, 'I':2, and 'P':3, where 'P' is used only as padding when creating the data loader. Text normalization is done by converting all text to lowercase. Values such as emojis or whitespace are not eliminated, although their corresponding IDs could be removed from the list of tokens and labels.

The German and Spanish dataset are downloaded from respectively (Muchafel, 2019) and (Basile et al., 2019). The German hate speech dataset consists of a train set of 180 examples, a validation set of 100 examples, and a test set of 100 examples. In the training set, there are 140 examples labeled as non-hate speech and 40 examples labeled as hate speech. In the validation set, there are over 100 non-hate speech examples and 40 hate speech examples. In the test set, there are 100 non-hate speech examples and 40 hate speech examples. The preprocessing phase is the same as the one applied to tasks A and B for tweet datasets. The ratio of positive to negative examples in the German hate speech dataset is as follows:

Set	Train	Val	Test
P	24.06 %	20.57 %	25.53%
N	75.94 %	79.43 %	74.47%

The Spanish hate speech dataset consists of a train set of 4500 examples, a validation set of 450 examples, and a test set of 1400 examples. In the training set, there are 2500 examples labeled as non-hate speech and 2000 examples labeled as hate speech. In the validation set, there are over 250 non-hate speech examples and 200 hate speech examples. In the test set, there are over 800 non-hate speech examples and 600 hate speech examples.

Set	Train	Val	Test
P	41.27 %	44.40 %	41.25%
N	58.73 %	55.60%	58.75%

5 Experimental setup and results

Plotting the length of tweets and news dataset on histograms, we fixed the 'max_length' for tokenization. For all tweet's tasks, we set the parameter to 256 because tweets reach 250 token's length in task A, B and with Spanish dataset. For task A and B, the tweets maximum length is 256 (see fig 3 and 4) while for news the lengths are lower than 110 tokens (fig 5).

For task C, the tweets maximum length is around

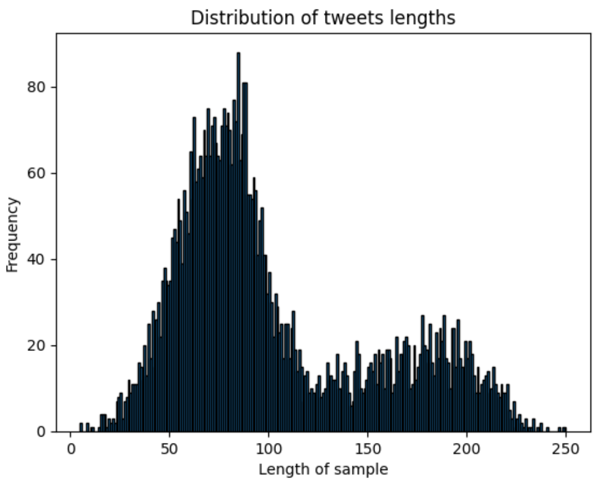

Figure 3: Task A - tweets

140 so max 'max_length' is 140 (see fig 6). In fig 7, Spanish tweets length is maximum 256 while for German is 130 (shown in fig 8). Despite this, we have set the max length at 256 to compare with the results of the other models. The Adam optimizer and BCEWithLogitsLoss are set as the optimizer and loss function, respectively, for training the model. BCEWithLogitsLoss is a loss function that applies the Sigmoid layer and binary cross-entropy loss function in a single function. We preferred it due to its numerical stability and faster convergence during training compared to BCELoss.

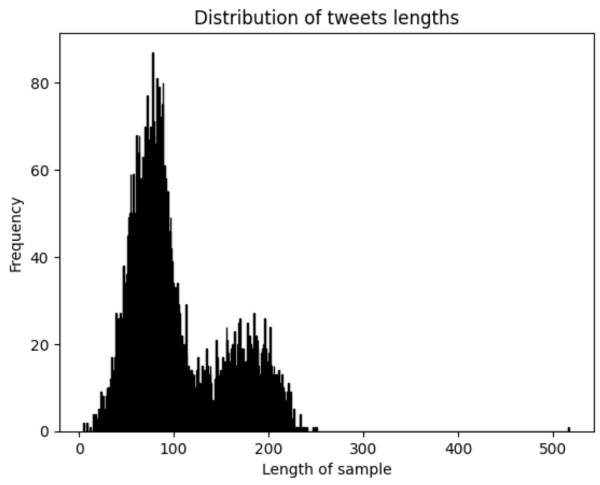

Figure 4: Task B - tweets

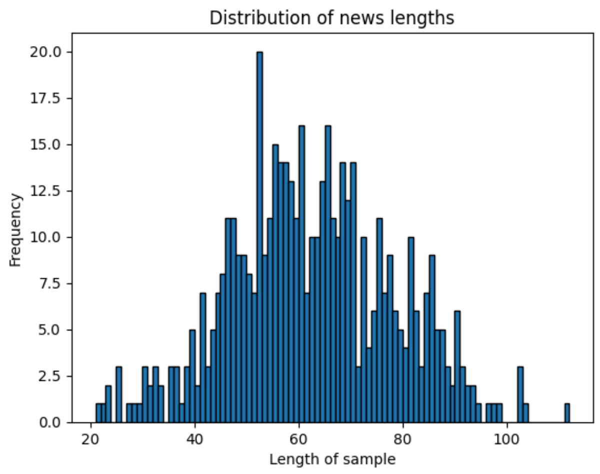

Figure 5: Task A/B - news

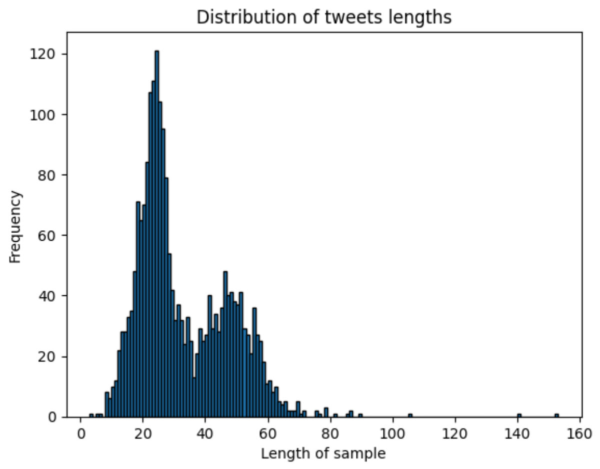

Figure 6: Task C - tweets

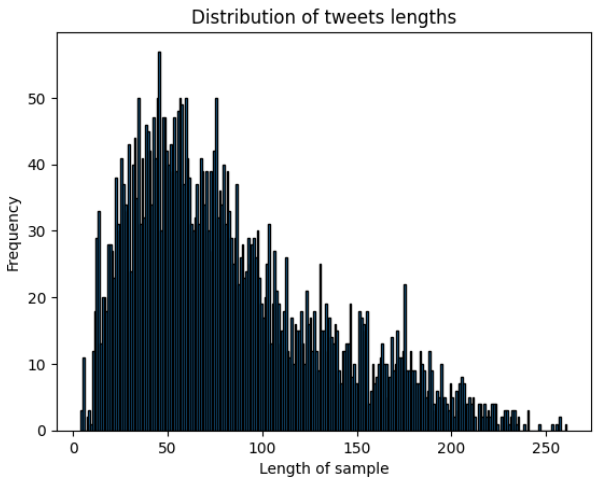

Figure 7: Spanish - tweets

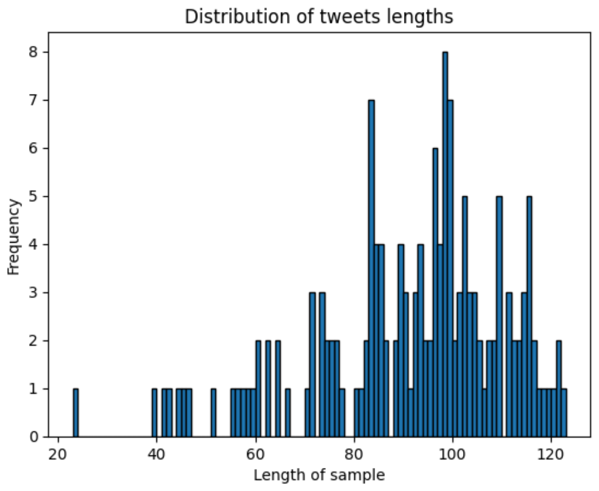

Figure 8: German - tweets

It is more numerically stable than using a plain Sigmoid followed by BCELoss. BCEWithLogitsLoss has a pos_weight parameter that is a weight of positive examples. The value of the pos_weight parameter can be chosen based on the negative/-positive class frequency in the training set. A good starting point could be to set the pos_weight to be the ratio of the negative over positive classes in the training set. To maximize precision, we want to minimize false positives. Therefore, we set the pos weight value to 1.5 for task A, where the positive class frequency is 40%.
During the evaluation phase, the float output value from the model is split by a threshold value of 0.5, where output sigmoid values greater than 0.5 are classified as 1 and values less than 0.5 are classified as 0.
The hyper-parameters for the model are chosen using the Grid Search technique. For task A and B, the hyper-parameter grid is made up of seeds, dropout rate, and batch size with a fixed learning rate of 1e-5 and pos_weight over 20 epochs. For task C, the hyper-parameter grid is made up of seeds, dropout rate, batch size, max length, and number of labels with a fixed learning rate of 1e-5 over 20 epochs. After analyzing the models' performance with the set of hyper-parameters over F1 score and val loss, we choose the optimal values to boost model performance. In the following table are shown the F1 score and val loss values for optimal hyper-parameters that maximize the F1 score and minimize the val loss:

	F1-score	Val loss
A	0.768	0.617
B	0.688	0.699
C	0.654	0.073

The settings were chosen based on the lowest value of the validation loss/F1 score ratio. The models were trained using the best hyperparameters selected with Grid-Search before the evaluation. The table below shows the final hyperparameter settings for each task.

	s	tl	lr	b	d	p	#l
A	42	256	1e-5	32	0.3	1.5	/
B	42	256	1e-5	32	0.8	1.25	/
C	12321	140	1e-5	64	0.2	/	4

s: seed, tl: tokenizer length, lr: learning rate, b: batch size, d: dropout rate, p: pos weight, #l: number of labels.
In multilingual extension of Task A:

	F1-score	Val loss
Spanish	0.698	0.954
German	0.821	0.486

The final hyper-parameters configuration in the table below:

	s	tl	lr	p	b	d
Spanish	256	12321	1e-5	1.4	32	0.5
German	140	12321	1e-5	2.8	64	0.2

s: seed, tl: tokenizer length, lr: learning rate, p: pos weight, b: batch size, d: dropout rate.
The chart plots the trend of training loss and validation loss over epochs before early stopping. In the case of task A, the graph shows that both the training loss and validation loss are decreasing, indicating that the machine learning model is learning and improving its performance by reducing the error between its predictions and the actual values. This enhances the model's ability to make accurate and generalizable predictions on new data.9. For

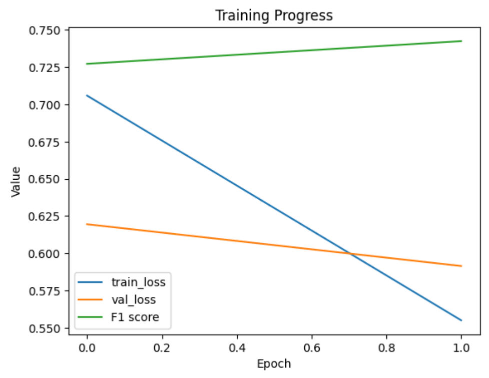

Figure 9: Training progress Task A

Task B the val loss and train loss decreases but val loss is higher than train loss in figure 10. For task C both the train loss and validation loss decrease and almost overlap, indicating that the model is effectively learning from the data and making accurate predictions on the training and evaluation data. This is a positive indication of the model's ability to fit the data and generalize to new data.11. Also for the Spanish tasks both the validation loss and train loss decrease and meet at a point. However, the validation loss remains consistently higher than the train loss.12. For German model, the train loss and val loss meet in epoch 9. 13 The model with lowest val loss is the model of task C with a val

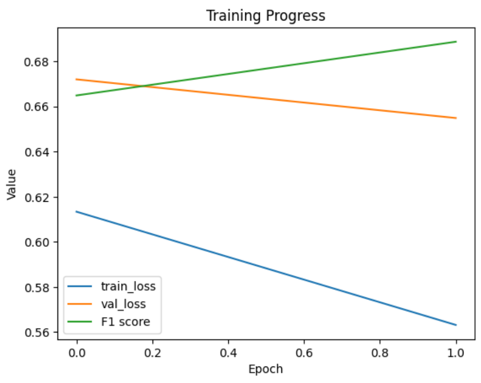

Figure 10: Training progress Task B

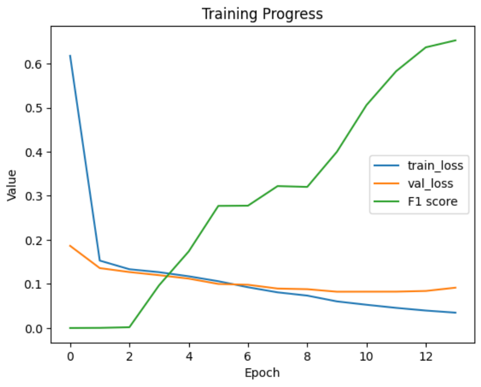

Figure 11: Training progress Task C

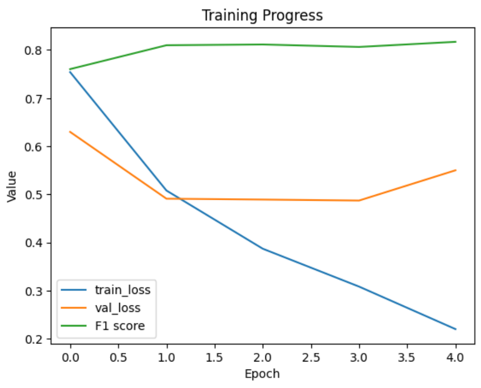

Figure 12: Training progress Spanish

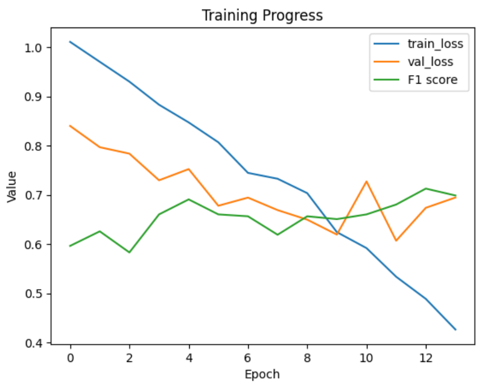

Figure 13: Training progress German

loss of 0.13.

Results of the Task A:

	ts	c	p	r	F1	vl
A	T	0	.878	.452	.579	.595
A	T	1	.623	.935	.748	.595
A	N	0	.754	.868	.807	.526
A	N	1	.681	.500	.576	.526

ts: train set, c: class, p: precision, r: recall, F1: F1-score, vl: val loss.

Results of the Task B:

	ts	c	p	r	F1	vl
B	T	0	.760	.675	.715	.614
B	T	1	.652	.741	.694	.614
B	N	0	.693	.956	.804	.665
B	N	1	.730	.217	.334	.665

ts: train set, c: class, p: precision, r: recall, F1: F1-score, vl: val loss.

Results of the Task C:

	ts	c	p	r	F1	vl
C	NU	1	0.626	0.505	0.559	.137
C	NU	2	0.686	0.587	0.633	.137

ts: train set, c: class, p: precision, r: recall, F1: F1-score, vl: val loss.

The multilingual extension:

	c	p	r	F1	vl
Spanish	0	0.843	0.7064	0.769	.699
Spanish	1	0.660	0.812	0.728	.699
German	0	0.789	0.824	0.806	.612
German	1	0.478	0.423	0.449	.612

ts: train set, c: class, p: precision, r: recall, F1: F1-score, vl: val loss.

The F1-score macro avg of each model.

	A	B	C	Sp	Ge
F1-score	0.673	0.705	0.596	0.748	0.627

The F1-score macro avg of Task A and B over news.

	A	B
F1-score	0.692	0.569

6 Discussion

Based on the results, it is clear that there are some differences between the performance of Task A and Task B models. Specifically, the Task A model has a lower loss over news compared to Task B, while

the Task B model has a lower loss over tweets. It is also evident that the F1 score for the positive label significantly drops in the news dataset for both Task A and Task B. Perhaps the difference in the balance of labels between the training and test sets for tweets is contributing to the performance issues, or it could be that the pos_weight hyperparameter is not optimally set and needs better tuning. It seems that the models perform better on datasets with a positive/negative label ratio that is more similar to the ratio in the training set.
In the multilingual extension, the Spanish model has a slightly higher loss compared to the German model. However, the F1 scores for the Spanish model are higher and more balanced compared to the German model, which results in a very low F1 score for the positive label. This could be due to the lower number of examples in the German dataset.
Analyzing the f1 score over classes for Task C, we can conclude that the model performs better in detecting inside tokens of NUs than begin tokens.
To improve the model's performance, could be usefull to make some changes to the pre-processing of tweet text. One approach could be to insert spaces within hashtags to better capture the meaning of the words, and to substitute emojis with their word equivalents to enhance interpretability.

7 Conclusion

In this project, we aimed to test some deep learning models to detect hate speech in social media, specifically Twitter. We trained and evaluated the model on three different tasks with varying complexities and data characteristics. We also performed a multilingual extension training it on Spanish and German.
While our models achieved good results overall, there were still some limitations that we encountered. We observed that the Spanish model performed the best, with higher F1 scores and more balanced performance on positive and negative labels, compared to the German and Italian models. Additionally we found that our Italian models performed better on sets with a positive/negative label ratio similar to that of the training set, indicating a need for more sophisticated sampling techniques to handle imbalanced datasets. Also we noted the importance of considering emoji and hashtag processing in the text pre-processing step, specifically for the NER task. Finally, the multilingual extension of the model posed some challenges, such as differences in language structure and word meaning.
Future improvements could include exploring more advanced techniques for data preprocessing and sampling, as well as exploring more effective methods for German extension like augmenting the number of examples in the dataset.

8 Links to external resources

Here the links of our GitHub repository:

- Task A, B and C dataset Github-ABC-ds
- Spanish dataset Gihub-Spanish-ds
- German dataset Github-German-ds

References

Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Manuel Rangel Pardo, Paolo Rosso, and Manuela Sanguinetti. 2019. Semeval 2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In *Proceedings of the 13th international workshop on semantic evaluation*, pages 54–63.

Sanguinetti Manuela, Comandini Gloria, Elisa Di Nuovo, Simona Frenda, MARCO ANTONIO Stranisci, Cristina Bosco, Caselli Tommaso, Viviana Patti, Russo Irene, et al. 2020. Haspeede 2@ evalita2020: Overview of the evalita 2020 hate speech detection task. In *Proceedings of the Seventh Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2020)*, pages 1–9. CEUR.

Bjoernross Muchafel. 2019. Iwg hatespeech public.