UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

Master's Degree in Computer Science
AI curriculum

Human Language Technologies: Hate Speech Detection with BERT and LSTM

Prof:

Attardi Giuseppe

Candidate:

Paterniti Barbino Niko
Mat: 638257

ANNO ACCADEMICO 2022/2023

Indice

Capitolo 1

Introduction

Online hateful content, or Hate Speech (HS), is characterised by some key aspects (such as virality, or presumed anonymity) which distinguish it from offline communication and make it potentially more dangerous and hurtful.Therefore, its identification has become a crucial mission in many fields.
From an NLP perspective, much attention has been paid to the topic of HS, together with all its possible facets and related phenomena, such as offensive/abusive language,its identification and so on.
The aim of this project is the building of an Hate Speech Detection system able to recognize if a given tweet or comment contains hateful content or not.
For this purpose we used the dataset provided by the Evalita-2020 challenge (haspeede2 task).
In the first section we will describe the prior work which helped me in developing this project.
In the second section the dataset used will be described.
The third section describes the architectures of the implemented models.
The fourth seection will present the results obtained also describing the evaluation methods used.
In the last section conclusions and possible ideas for future improvements will be presented.

Capitolo 2

Prior Work

The first step into the development of this project was researching and understanding the task.
After reading some papers i figured that the best models for approaching the task of Hate Speech Detection are:

- Long Short-Term Memory (LSTM)
- bi-directional LSTM
- Transformer based methods:(Small BERT, BERT, AlBERT)

Before the introduction of the Bert model, the bidirectional LSTM based models, were among the most used model to approach NLP problems.
Recent studies proved that Bert based models are extremely effective in facing NLP leading into a decrease in popularity of LSTM based models.
RNN are slow to train and to infer, this is because words are processed one at a time, so longer sentences take longer time. They dont truly understand the context of a word, normally these networks learn from words that come before it but the context depends from both the words that come before and after it.
Even bi-directionals models (BILSTM) suffer from this since it simply learns left right and right to left context separately and then concatenates them so it isn't learning the true meaning of the word, so at the end the true context might be slightly lost.
Transformers can solve both problems, the Transformer architecture introduced by Vaswani et al. in the paper "Attention is All You Need" in 2017, has become a foundational model for natural language processing (NLP) tasks and beyond.
One of the main advantages of BERT over Bi-LSTM in the context of NLP is its ability to capture contextual information effectively through bidirectional attention mechanisms. This bidirectional processing allows BERT to consider both the left and right context of each word in a sentence simultaneously. The significance of this capability lies in its ability to capture rich contextual embeddings for each word, taking into account the entire surrounding context.

The bidirectional attention in BERT enables the model to understand the meaning of a word in the context of the entire sentence, making it more effective in capturing long-range dependencies and relationships between words. This is crucial for tasks such as language understanding, sentiment analysis, and various other NLP applications where the meaning of a word often depends on its context within the broader text. BiLSTM, on the other hand, processes input sequentially, and while it considers information from both directions, it doesn't capture global context as effectively as BERT and lacking the attention mechanism present in transformers, they are less effective at capturing relationships between distant words.
While other advantages like parallelization are important, the ability of BERT to capture contextual information bidirectionally stands out as a key factor contributing to its superior performance in a wide range of NLP tasks compared to architectures like BiLSTM.
So i decided to approach the Hate Speech Detection task comparing AlBERTo (a pretrained BERT language understanding model for the Italian Language) and the bi-directional LSTM model in order to provide interesting comparisons between them.

Capitolo 3

Dataset

The dataset used in this project is the one provided by the EVALITA HaSpeeDe2 competition organizers. In particular, the entire dataset is split into one Training Set composed of tweets and two test sets: an in-domain (based on tweets) and a smaller out-of-domain (based on newspaper phrases) test set. Overall, the Training Set includes 6,839 Italian tweets distributed as in Table 3.1. The dataset includes texts targeting minority groups such as Immigrants, Muslims and Roma communities, whose relative social problems constantly feed the public and political debate triggering hate speech

- Task A - Hate Speech Detection (MainTask): binary classification task aimed at determining the presence or the absence of hateful content in the text.

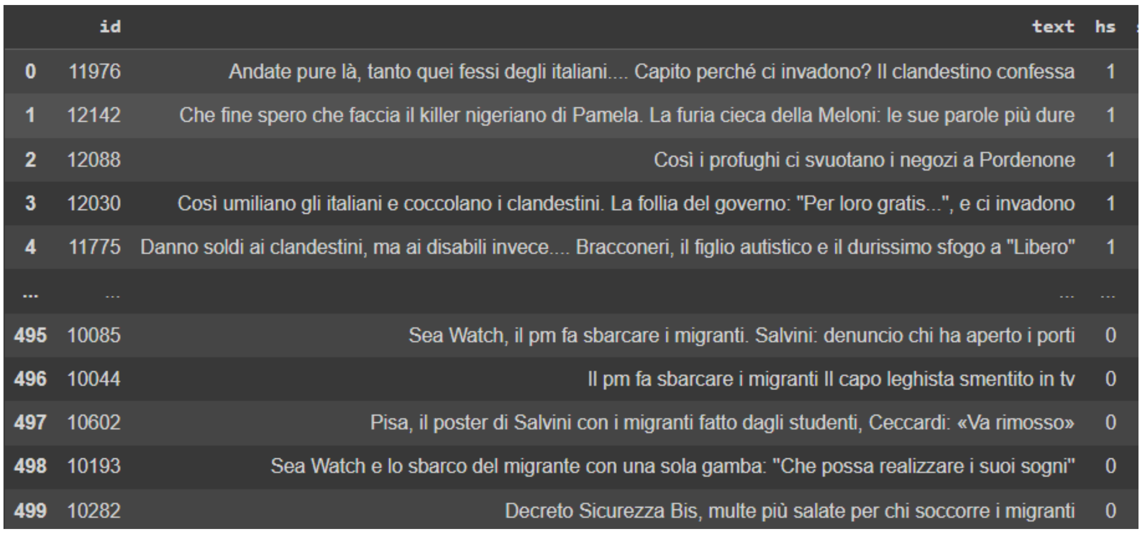
Task A	HS	NOT HS	TOT.
Train	2766	4073	6839
Test Tweets	622	641	1263
Test News	181	319	500

Tabella 3.1: Dataset distribution

3.1 Data format

The dataset is provided in a tab-separated values (TSV) file including ID, text, HS and class (0 or 1). Mentions and URLs were replaced with @userand URL placeholders. The following figure shows some annotation examples.

- id: a number that substitutes the original tweet ID
- text: the tweet
- hs: the hate speech class: 1 if the tweet contains hateful content, 0 otherwise



	id	text	hs
0	11976	Andate pure là, tanto quei fessi degli italiani.... Capito perché ci invadono? Il clandestino confessa	1
1	12142	Che fine spero che faccia il killer nigeriano di Pamela. La furia cieca della Meloni: le sue parole più dure	1
2	12088	Così i profughi ci svuotano i negozi a Pordenone	1
3	12030	Così umiliano gli italiani e coccolano i clandestini. La follia del governo: "Per loro gratis...", e ci invadono	1
4	11775	Danno soldi ai clandestini, ma ai disabili invece.... Bracconeri, il figlio autistico e il durissimo sfogo a "Libero"	1
...	...	...	...
495	10085	Sea Watch, il pm fa sbarcare i migranti. Salvini: denuncio chi ha aperto i porti	0
496	10044	Il pm fa sbarcare i migranti Il capo leghista smentito in tv	0
497	10602	Pisa, il poster di Salvini con i migranti fatto dagli studenti, Ceccardi: «Va rimosso»	0
498	10193	Sea Watch e lo sbarco del migrante con una sola gamba: "Che possa realizzare i suoi sogni"	0
499	10282	Decreto Sicurezza Bis, multe più salate per chi soccorre i migranti	0

Figura 3.1: Dataset example

3.2 Data Preparation

A Tweet is a text message with a maximum length of 280 characters. It may contain elements such as hashtags, mentions, links and emoticons.
Due to that the data preprocessing phase has been carried out implementing a series of functions with the aim of modifying a tweet to eliminate useless and ambiguous elements in order to standardize it:

- Tags removal
- Special characters transformation (for example & become e)
- Conversion of disguised bad words (at times in order to avoid to be censured, bad words are written in a disguised way using special characters inside it so that they are not detectable).
- URL removal
- misspelling adjustment (removing nearby equal vowels)
- translation of emoticons
- replacement of abbreviations with the respective word
- laughs removal
- lower case tweet transformation

An example of a tweet extracted from the dataset is shown below:
"user La società multirazziale... #migranti #profughi #rom URL"
As a result the tweet becomes:
l̈a società multirazziale migranti profughi rom

Capitolo 4

Architecture

In the context of Hate Speech Detection, NLP (Natural Language Processing) allows us to teach machines to understand and analyze human language. Since Hate Speech can be subtle and context dependant, solving this task would be really challenging for traditional rule based systems.
NLP enables algorithms to recognize patterns, context and nuances in language, helping in distinguishing between ordinary speech and potentially harmful content. NLP provides a powerful tool in the context of effectively detecting Hate Speech in an automated manner.

4.1 Models

I decided to approach the task by analyzing and comparing two different models:

- **ALBERTo**: pre-trained language model built upon the BERT architecture but adapted and fine-tuned specifically for Italian language understanding. It's trained on a large corpus of Italian text, enabling it to capture the intricacies and nuances of the language.

- **BiLSTM** : a type of recurrent neural network (RNN) architecture used in natural language processing tasks. It's an extension of the traditional LSTM (Long Short-Term Memory) network.

4.1.1 BERT

The Transformer model consists of an encoder and a decoder, both composed of multiple identical layers. While the original Transformer was designed for sequence-to-sequence tasks (e.g., machine translation), by stacking the encoders we get BERT (Bidirectional Encoder Representation from Transformers), so BERT is a stack of transformer encoders.
The core innovation of the Transformer is the self-attention mechanism, instead of processing words in a sequence, self-attention allows the model to focus on different

parts of the input sequence simultaneously. Each word in the input sequence can attend to all other words, and the importance of each connection is dynamically calculated based on learned attention weights.
The self-attention mechanism is extended to multi-head attention in the Transformer. In multi-head attention, the model uses multiple sets of attention weights to capture different aspects of the relationships between words. Each head operates independently, and their outputs are concatenated and linearly transformed to produce the final output.
Since the Transformer does not inherently understand the order of the input sequence, positional encoding is added to the input embeddings. This encoding provides information about the relative or absolute position of each token in the sequence, allowing the model to consider the order of the input data.
In the case of the Transformer used for pre-training models like BERT, only the encoder layers are used. The encoder processes the input sequence, and the output from the final encoder layer is often used as the contextualized representation of the input.
Transformers are highly parallelizable, making them efficient for training on large datasets and powerful hardware.
We can pre-train BERT to understand language and fine-tune it to learn specific task, so the training of BERT is done in two phases.

- **TRAINING PHASE:**
  The goal of pretraining is making bert learn what is language and what is context. BERT learns language by training on two unsupervised task:
  - **Masked Learning**: (helps bert learn bidirectional context within a sentence)
  - **Next following sentence**: (takes a sentence and determines in the next sentence actually follows the first(helps bert understand context withing different sentences)
- **FINE-TUNING PHASE:**
  Now bert can be trained on very specific NLP tasks. We just need to learn the output model parameters and the rest of the model parameters are just slightly fine tuned so faster training. So depending on what task we want to solve we just need to change the output layers and train with a specific dataset.

In the pre-training phase, the input is a set of sentences with a set of words being masked, each token is a word, we convert each of these words into embeddings using pretrained embeddings.
C is the output of the next following sentence prediction, it outputs 1 if sentence B follows A, 0 otherwise.The outputs of the masked learning instead are word vectors, so the number of word vectors we gave in input is the same as the number we get in output.

To generate the embeddings for the words token inputs, the initial embeddings are constructed from 3 vectors:

- Token Embeddings (pretrained embeddings)
- Segment Embeddings (sentence number encoded into a vector)
- Position Embeddings (position of word that is encoded into a vector)

Adding those 3 together we get an embedding vector that we use as input to bert. The segment and position embeddings are required for temporal ordering since all these vectors are fed in simultaneously into bert and language models need this ordering preserved.
The output is a a binary value c and a bunch of word vectors. All of the word vectors have the same size and are generated simultaneously. We need to take each word vector and pass it into a fully connected layer output with the number of neurons being the same as the number of tokens in the vocabulary and apply a *softmax* activation so that we can convert a word vector to a distribution and the actual label of this distribution would be a one hot encoded vector for the actual word.
We compare this two distributions and the train the network using the cross entropy loss.The loss only considers the prediction of the masked words ignoring all the other words output by the network by doing that we increase context awareness because we focus more on predicting those masked terms.

4.1.2 ALBERTo

In the context of hate speech detection in NLP, ALBERTo can be used as a powerful tool to understand and analyze Italian-language text for potentially harmful content. Its pre-trained representations can be fine-tuned on hate speech datasets, allowing it to learn the specific patterns and features associated with hate speech in Italian.
The advantage of using pre-trained models like ALBERTo is that they capture a broad understanding of language from a diverse range of data, and fine-tuning allows them to specialize for specific tasks like hate speech detection in a particular language. This approach often yields better performance than training a model from scratch on a limited dataset.
AlBERTo was originally trained using only the masked learning technique because it is used with Tweets data, and we do not have cognition of a flow of tweets as it happens in a dialogue that has questions and answers.
Because of this, AlBERTo is not suitable for question answering tasks but it is suited for classification and prediction tasks.
BERT models implementation consist of two phaes:

- **Pre-training:** the model is trained on unlabeled data over different pre-training tasks

- **Fine-tuning:** the model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.In order to achieve good performances the fine-tuning has been performed on the ALBERTo model.

This architecture creates a feed-forward neural network on top of the pre-trained BERT model. The BERT model extracts contextualized representations from the input text, and the feed-forward classifier processes these representations for the specific classification task.
The model architecture consists of:

- **Input Layer**
- **Hidden Layer 1:** Linear + ReLU + Dropout
- **Hidden Layer 2:** Linear + ReLU + Dropout
- **Output Layer:** Linear

Each layer is composed of a linear transformation, a ReLU activation, and a dropout operation.

```python
class BertClassifier(nn.Module):
    """Bert Model for Classification Tasks.
    """
    def __init__(self, freeze_bert=False):
        """
        @param    bert: a BertModel object
        @param    classifier: a torch.nn.Module classifier
        @param    freeze_bert (bool): Set `False` to fine-tune the BERT model
        """
        super(BertClassifier, self).__init__()
        # Specify hidden size of BERT, hidden size of our classifier, and number of labels
        D_in, Hidden_1, Hidden_2, D_out = 768, 256, 64, 2

        #Instantiate BERT model
        #Italian BERT model with an uncased vocabulary,
        #12 layers, a hidden size of 768, and 12 attention heads.

        self.bert = BertModel.from_pretrained('m-polignano-uniba/bert_uncased_L-12_H-768_A-12_italian_alb3rt0')

        #Instantiate a sequential feed-forward classifier
        self.classifier = nn.Sequential(
            nn.Linear(D_in, Hidden_1),
            nn.ReLU(),
            nn.Dropout(0.7), #dropout layer with a dropout probability of 0.7(helps prevent overfitting)
            nn.Linear(Hidden_1, Hidden_2),
            nn.ReLU(),
            nn.Dropout(0.7),
            #Final layer D_out=2 because we have 2 classes in our classification task.
            nn.Linear(Hidden_2, D_out)
        )

        # Freeze the BERT model
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False
```

4.1.3 BiLSTM

The Bidirectional Long Short-Term Memory (BiLSTM) model architecture for hate speech detection involves the use of Bidirectional LSTMs, which are a type of recurrent neural network (RNN).

- **Forward and Backward Processing:** In a standard LSTM, information flows from left to right, capturing dependencies from the past. However, a BiLSTM processes the input sequence in both forward and backward directions simultaneously. This bidirectional processing helps the model to understand the context from both the past and future.

- **Memory Cells:** LSTMs have memory cells that can store and retrieve information for long-term dependencies. This is crucial for hate speech detection as understanding the context of certain words may require information from both directions in the sequence.

- **Gates:** LSTMs have gates that control the flow of information. These gates include input gates, forget gates, and output gates, allowing the model to selectively update and use the information in the memory cells.

- **Embedding Layer:**
  Word Representations: The input words are typically represented as embeddings, which are dense vector representations. These embeddings capture semantic relationships between words.

- **Bidirectional Aspect:**
  Forward and Backward Hidden States: The forward and backward hidden states from the BiLSTM are concatenated to create a comprehensive representation of the input sequence. This concatenated representation encodes both the past and future context of each word.

- **Output Layer:**
  Classification Layer: The final hidden states are often fed into a classification layer (commonly a softmax layer) to predict whether the input sequence contains hate speech or not. This layer is trained using labeled data, and the model learns to associate patterns in the input sequence with hate speech labels.

The BiLSTM architecture is characterized by:

- **lstml1_in:** Input layer for the text data with a shape of (max_length, 128). It assumes the input sequences have a maximum length of max_length and each word is represented by a vector of size 128.

- **lstml1_bd1:** Bidirectional LSTM layer with return_sequences=True to return the full sequence of outputs. This layer processes the input sequences bidirectionally.

- **lstml1_bd2:** Another Bidirectional LSTM layer without return_sequences=True. It processes the output sequence from the previous layer and aggregates information bidirectionally.
- **other_in:** Input layer for other features with a shape of (6,). It assumes there are 6 additional features.
- **lconcat:** Concatenation layer that concatenates the output from the last bidirectional LSTM layer and the other features.
- **dense1_layer, dense2_layer, dense3_layer:** Dense layers with ReLU activation and L2 regularization. They serve as additional processing layers after concatenation.
- **lstml1_out:** Output layer with a sigmoid activation function for binary classification.

The model is designed to handle two types of inputs: sequential text data ("text") and additional features ("other") that will later be explained in the *Experiments* section. The bidirectional LSTM layers capture sequential patterns from the text data, and the additional features are incorporated through concatenation and subsequent dense layers. The final output is a binary classification using a sigmoid activation function

```

def get_model(hparams):
  lstml1_in = tf.keras.layers.Input(name="text", shape =(max_length,128,))
  lstml1_bd1 = tf.keras.layers.Bidirectional(LSTM(hparams[HP_NUM_UNITS], dropout = hparams[HP_DROPOUT], return_sequences=True))(lstml1_in)
  lstml1_bd2 = tf.keras.layers.Bidirectional(LSTM(hparams[HP_NUM_UNITS], dropout = hparams[HP_DROPOUT]))(lstml1_bd1)

  other_in = tf.keras.layers.Input(name="other", shape =(6,))

  lconcat = tf.keras.layers.Concatenate(axis=1)([lstml1_bd2, other_in])
  dense1_layer = Dense(256, activation="relu", kernel_regularizer=tf.keras.regularizers.l2(hparams[HP_L2]))(lconcat)
  dense2_layer = Dense(64, activation="relu", kernel_regularizer=tf.keras.regularizers.l2(hparams[HP_L2]))(dense1_layer)
  dense3_layer = Dense(32, activation="relu", kernel_regularizer=tf.keras.regularizers.l2(hparams[HP_L2]))(dense2_layer)

```

```
lstml1_out = tf.keras.layers.Dense(1, activation='sigmoid')(
 dense3_layer)

model = tf.keras.Model(inputs = [lstml1_in, other_in],
 outputs = lstml1_out)

model.summary()

return model
```

Capitolo 5

Experiments

5.1 ALBERTo

While training the model i found the issue of understanding how many epochs of training were needed in order to achieve good results.
I tried different ways of fine-tuning:
Initially i tried the full training of the model, then i switched to the training of only the final classification dense part, the results of the test were not great, so i decided to initially fine-tune ALBERTo on the full model (3 train epochs) and then i tried training on the final dense connected part (7 train epochs) keeping the rest of the parameters of the model constant. This choice came from different reasons:

- BERT, being a pre-trained language model, has learned valuable contextual representations from a large corpus during pre-training. By training on the entire model initially, we allow the model to adapt to the specific task and dataset, leveraging the knowledge encoded in BERT's pre-trained weights.

- The initial training on the entire model allows the network to adapt to task-specific features present in the dataset. This is particularly useful when the pre-trained BERT model needs to be fine-tuned for a specific downstream task, such as hate speech detection.

- Freezing the BERT part after initial training helps in preventing "catastrophic forgetting." When having limited task-specific data, freezing the pre-trained BERT layers helps retain the valuable linguistic knowledge it has acquired. If we were to keep updating the BERT weights with task-specific gradients over a small dataset, the model could forget the general language understanding it gained during pre-training.

- Training the entire BERT model can be computationally expensive. By pre-training the BERT layers and then freezing them, we can reduce the number of parameters that need to be updated during the subsequent training phase. This helps in faster training and requires less computational resources.

- The pre-trained BERT layers are considered to be more stable and generalizable across various language understanding tasks. By freezing them, we ensure that the model relies on these general features while allowing the dense part to specialize for the specific task.
- In many real-world scenarios, obtaining labeled data for a specific task is challenging and limited. Training the entire model with limited data might lead to overfitting. By freezing BERT and training only the dense part, we can fine-tune the model with a smaller amount of task-specific data.
- When training multiple models or using an ensemble, freezing BERT and training only the dense part can be especially beneficial. It ensures that the ensemble members focus on task-specific patterns while sharing a common, pre-trained linguistic understanding

In summary, training on the entire model and then freezing BERT is a practical strategy that balances leveraging pre-trained knowledge with task-specific adaptation and, as in this case, it often leads to better generalization on downstream tasks with limited data.

5.2 BiLSTM

During my experiments i found out that the model performed decently but with the aim of doing better i tried adding to the training set additional fetures useful to help the model in understanding better if the text is hateful or not.
The model in this way in fact learns to make predictions based on the information from both sets of features during the training process. This can be beneficial if the additional features provide valuable information for several reasons:

- **Capturing Linguistic Patterns:**
  - Caps Lock Words: Hate speech may be associated with aggressive or emphatic language, which might include the excessive use of capital letters.
  - Exclamation Points: Hate speech may contain heightened emotions, and the excessive use of exclamation points can be an indicator.
- **Identifying Offensive Language:**
  - Number of Bad Words: Explicitly offensive words or slurs are common in hate speech. Counting the occurrences of such words can provide valuable information.
- **Contextual Information:**
  - Sentence Length: Hate speech might be characterized by unusually short or long sentences. Extracting sentence length can capture such patterns.
  - Punctuation Usage: Certain types of punctuation (e.g., multiple question or exclamation marks) may be indicative of hate speech.

- **Model Generalization:**
  - Additional Features for Improved Generalization: Introducing features beyond raw text can help the model generalize better across different datasets or domains. These features can act as complementary information to the semantic content of the text.

- **Model Explainability:** - Interpretability: Features like the number of caps lock words or exclamation points can provide interpretable signals to understand why a particular prediction was made. This can be crucial for transparency and accountability.

- **Handling Data Imbalance:**
  - Balancing the Dataset: Hate speech detection datasets are often imbalanced, with fewer instances of hate speech. Extracting features related to offensive language or aggressive tone can help address this imbalance and provide more context to the model.

- **Behavioral Patterns:**
  - Stylometric Features: Extracting stylometric features (e.g., writing style, use of emojis) can capture unique behavioral patterns associated with hate speech

I also experimented using two types of models:

- Models with a pre-trained embedding layer
- Models without a pre-trained embedding layer

The results of the experiments showed that the models with the pre-trained embedding layer performed better in comparison to those without it, so i decided to consider only the better performing models.

I tested different configuration of the network each with a different number of bidirectional LSTM layers and a different configuration for the dense part.
Considering that a bigger and more complex model doesn't guarantee the best possible solution the models chosen are composed by:

- 2 bidirectional LSTM layers
- 3 dense layers of 256, 64 and 32 units
- Dropout, L2 regularization (to avoid overfitting/reduce complexity)

I then performed a grid search on the following parameters:

- Number of units (in the bidirectional LSTM layer): 64, 128, 256, 512
- dropout (in the bidirectional LSTM layer): 0, 0.2, 0.4, 0.6, 0.8
- L2 regularization on the desne layer 0.0, 0.0002, 0.0004, 0.0006, 0.0008

In the end, the first four selected models parameter were used in the training of 4 new network with k-fold cross validation, setting k equal to 5

Tabella 5.1: Best models

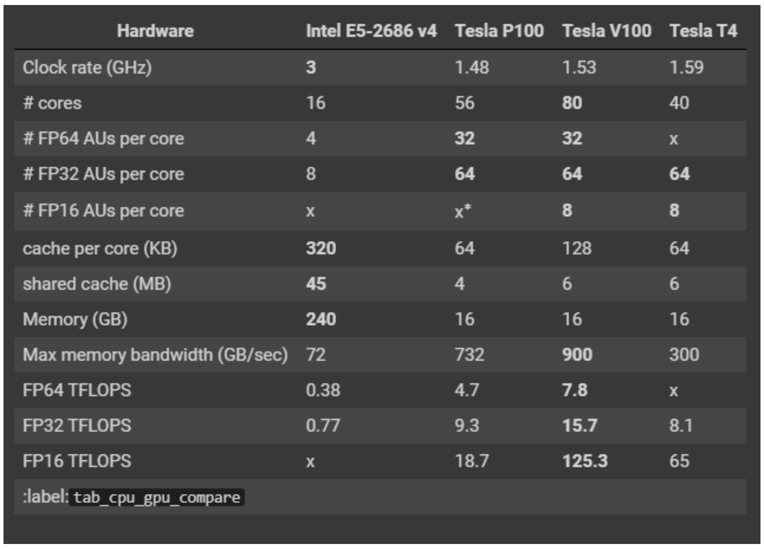
Units	**Dropout**	**L2 regularization**
64	0.2	0.0
64	0.4	0.0
128	0.6	0.0
256	0.6	0.0002

Capitolo 6

Results

The results as expected showed a better performances with the BERT models when compared to the BiLSTM model. Still the gap between the two models is not very huge, this is probably derived from the fact that:

- The training set requires more data
- Computational limits of Colab's notebooks free version.



Hardware	Intel E5-2686 v4	Tesla P100	Tesla V100	Tesla T4
Clock rate (GHz)	**3**	1.48	1.53	1.59
# cores	16	56	**80**	40
# FP64 AUs per core	4	**32**	**32**	x
# FP32 AUs per core	8	**64**	**64**	**64**
# FP16 AUs per core	x	x*	**8**	**8**
cache per core (KB)	**320**	64	128	64
shared cache (MB)	**45**	4	6	6
Memory (GB)	**240**	16	16	16
Max memory bandwidth (GB/sec)	72	732	**900**	300
FP64 TFLOPS	0.38	4.7	**7.8**	x
FP32 TFLOPS	0.77	9.3	**15.7**	8.1
FP16 TFLOPS	x	18.7	**125.3**	65
:label: `tab_cpu_gpu_compare`				

Figura 6.1: **Colab notebooks hardware comparison**

Probably a bigger hyperparameters grid search and a bigger dataset would have allowed to enlighten more the performance gap between the two models. This is also backed by the fact the when researching the results of the various haspeede competitions i found out that usually the best results add a "silver" dataset of thousands of hate speech tweets, proving that a model such as ALBERTo requires a huge amount of data to perform at its best.

6.1 F1 Score

As evaluation metric we decided to use the F1 score because the accuracy metric computes how many times a model made a correct prediction across the entire dataset, but this can be a reliable metric only if the dataset is class-balanced; that is, each class of the dataset has the same number of samples.
Nevertheless, real-world datasets are heavily class-imbalanced, often making this metric unviable.
The F1 Score is an alternative machine learning evaluation metric that assesses the predictive skill of a model by elaborating on its class-wise performance rather than an overall performance as done by accuracy. F1 score combines two competing metrics- precision and recall scores of a model, leading to its widespread use in recent literature.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

$$\text{Recall} = \frac{TP}{TP + FN} \qquad = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figura 6.2: Precisiom, Recall, F1 Score

6.2 BiLSTM

Tabella 6.1: BiLSTM Results

Configuration	Training Error	Vaidation Error	F1
64-0.2-0.0	0.2896	0.5621	0.75
64-0.4-0.0	0.4502	0.5142	0.76
128-0.6-0.0	0.4826	0.5843	0.73
256-0.6-0.0002	0.4861	0.5221	0.75

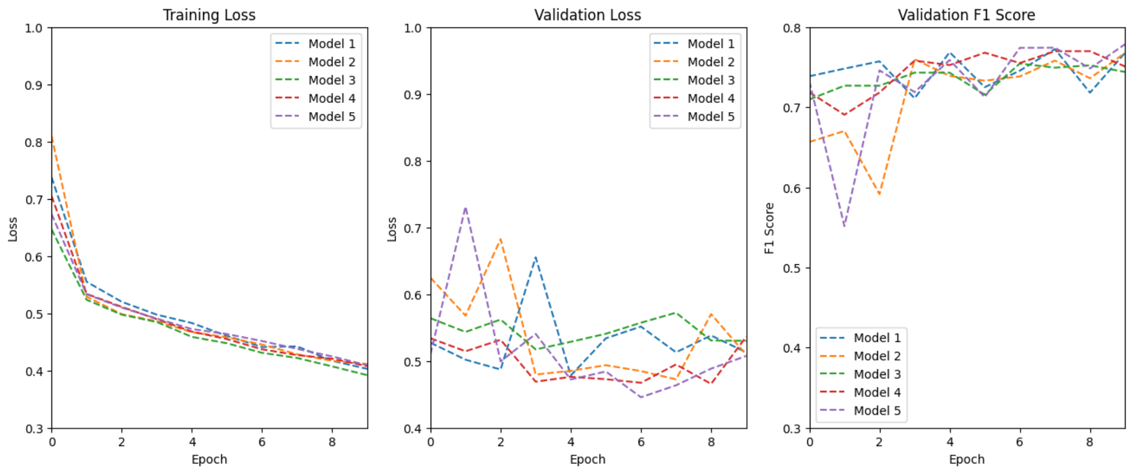
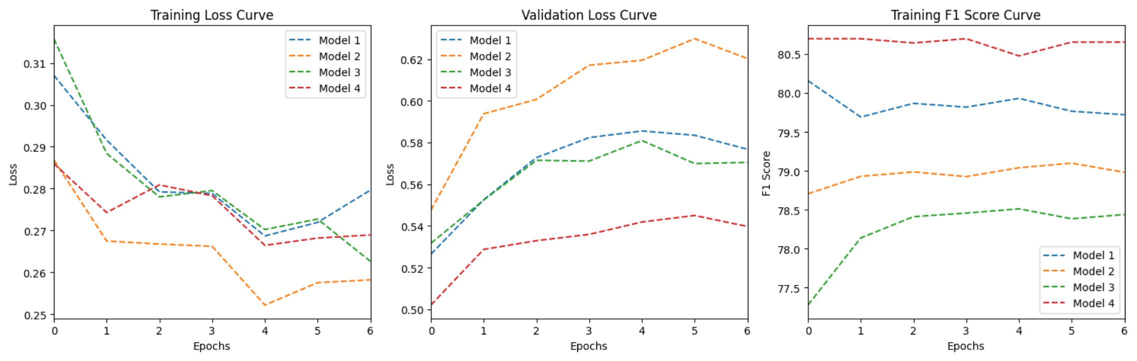


Figura 6.3: TR Loss, Validation Loss, F1 Score (64 - 0.4 - 0.0)

6.3 ALBERTo



6.4 Final results

I compared my results to the official results of HaSpeeDe 2 for Task A, ranked by the F1 score.
We can see that the Bert model performed better than the LSTM but still the gap is not so huge, as previously mentioned in my hypotesis about why this occurs.

Tabella 6.2: Final Results TWITTER DATA

Ranking	**Team**	**Model**	**F1 Score**
1	TheNorth_2	/	0.8088
2	TheNorth_1	/	0.7897
3	CHILab_1	/	0.7893
6	**Niko P.**	**ALBERTo**	**0.7697**
...	...	...	...
9	Jigsaw_al	/	0.7681
10	**Niko P.**	**BiLSTM**	**0.7623**
...	...	...	...
20	Baseline_SVC	/	0.7212
28	Baseline_MFC	/	0.3366

Tabella 6.3: Final Results NEWS DATA

Ranking	**Team**	**Model**	**F1 Score**
1	CHILab_1	/	0.7744
2	UO_2	/	0.7314
3	Montanti_1	/	0.7256
4	**Niko P.**	**ALBERTo**	**0.725625**
...	...	...	...
7	**Niko P.**	**BiLSTM**	**0.6868**
...	...	...	...
20	Baseline_SVC	/	0.621
28	Baseline_MFC	/	0.3894

Capitolo 7

Conclusions

Hate speech detection is a challenging task due to the dynamic and evolving nature of language. Hate speech can take various forms, and new expressions and slurs constantly emerge, making it difficult for algorithms to keep up.
Context is another hurdle, understanding the context in which certain words or phrases are used is crucial for accurate detection. The same words may be used differently in different contexts, and distinguishing between harmless expressions and actual hate speech requires a deep understanding of the surrounding text.
Additionally, cultural and regional variations in language make hate speech detection more complex. What may be considered offensive in one culture might not be in another. Algorithms need to be sensitive to these nuances to avoid misidentifying harmless content.
Moreover, the imbalanced distribution of hate speech data poses a challenge. Hate speech is often a minority class compared to non-hateful content, leading to imbalanced datasets. This can result in models that are biased towards the majority class and less effective at detecting hate speech.
Lastly, the subjective nature of hate speech adds to the difficulty, different people may have varying opinions on what constitutes hate speech, making it challenging to create a universal definition that algorithms can follow accurately.
Despite these challenges, ongoing research and advancements in natural language processing aim to improve the accuracy of hate speech detection systems.
Thanks to this project i understood the importance and the challenges posed by this task, the comparison with the BiLSTM model was also really interesting because even if the BERT models nowdays provide better performances, the results showed that BiSTM are still good models when facing Hate Speech Detection tasks.
As future studies it would be interesting to analyze the behaviour of other models such as the previously mentioned CNN (which allows to visualize sentences like an image).

Bibliografia

[1] EVALITA 2016-2020 (haspeede2)
http://www.di.unito.it/ tutreeb/haspeede-evalita20/index.html

[2] "AlBERTo the first italian BERT model for Twitter languange understanding"
https://github.com/marcopoli/AlBERTo-it

[3] Vitalflux, "Hate Speech Detection using Machine Learning"
https://vitalflux.com/hate-speech-detection-using-machinelearning/#:˜:text=Convolutional%20neural%20networks%20(CNN)%2C, detection%20using%20deep%20learning%20models.

[4] Hind Saleh Aree, Alhothali Kawthar, Moria "Detection of Hate Speech using BERT and Hate Speech Word Embedding with Deep Model"
https://arxiv.org/ftp/arxiv/papers/2111/2111.01515.pdf

[5] https://github.com/msang/haspeede/blob/master/2020/HaSpeeDe2020_Task_guidelines.pdf

[6] Rohit Cundu "F1 Score in Machine Learning: Intro & calculation"
https://www.v7labs.com/blog/f1-score-guide