NLP and AI-Powered Hate Detection: Unveiling the Secrets of Smart Algorithms to Combat Online Hatred

Text Mining and Natural Language Processing Exam 2022-2023

Moro Mattia [502259],Colosio Giacomo [509653]

14 July 2023

1 Introduction

1.1 Hate speech detection Introduction

Hate speech, is a phenomenon that threatens peaceful coexistence and respect for human rights in modern societies. It involves written expressions that incite hatred, violence, or discrimination toward people or groups on the basis of their ethnic, religious, political, sexual, or other affiliation. The spread of social networks has changed and as data show amplified the problem of hate speech, making it easier to produce and spread to a wide audience by giving the possibility of remaining anonymous and behind a screen. For many people, hate speech on social media can have serious and lasting consequences on the psychological and social well-being of its victims. Fortunately, there is also a potential solution to the problem, thanks to new natural language processing (NLP) and artificial intelligence (AI) technologies that can help detect and counter hate speech automatically and effectively. In particular, this project project aims to explore the challenges and solutions of hate detection in social media such as Twitter through the use of NLP and AI. Thus, with the aim of creating a program to automatically recognize messages that contain hate speech.

1.2 Hate speech detection, Artificial intelligence and NLP

Hate speech detection is a branch of sentiment analysis, one of the most widespread and relevant applications of natural language processing (NLP) and artificial intelligence (AI) in the field of text analysis. It involves identifying and classifying the emotions and opinions expressed in a text, which can be positive, negative or neutral as in the case of this project , or belong to more specific categories

such as joy, anger, disgust, sadness, fear and surprise. In particular sentiment analysis relies on machine learning and deep learning techniques, which consist of training mathematical models that can learn from labeled data. In supervised learning (supervised learning), labeled data consists of pairs of desired inputs and outputs, such as a text and its polarity. The model is trained to predict the correct output for each input, minimizing the error between the prediction and the true value. Sentiment analysis primarily uses supervised learning techniques, as it requires associating each text with a label that represents its sentiment. In particular Sentiment analysis is a task that in the world of machine learning and deep learning is commonly referred to as text classification, one of the oldest and most studied challenges in the field of NLP, with origins dating back to the 1950s. However, the development of sentiment analysis only exploded with the spread of social media, which provided a large amount of subjective and questionable texts for analysis

2 Data

2.1 Input Description

The data used for the project are the same collected from a scientific article published in 2017, which analyzed a sample of around 25,000 English-language tweets containing racist, sexist or homophobic terms from Hatebase.org. After that, through the online platform called CrowdFlower, each tweet was assigned to a label of one of these three categories: hate speech, offensive language or neither. In particular:

- Hate speech is defined as language that expresses hatred or violence towards a group or individual based on their ethnic, religious, political, sexual or other affiliation.
- Offensive language is defined as language that uses vulgar or scurrilous words to offend or insult someone or something.
- The neither category includes tweets that don't fit into the other two categories, such as those that use racist, sexist, or homophobic terms in an ironic, sarcastic, or self-referential way.

The data is stored as a CSV file and as a pandas dataframe where each element contains 6 columns:

1. Count: number of CrowdFlower users who annotated each tweet (minimum 3, sometimes multiple users annotated the same tweet when the reviews were considered unreliable by CF).
2. Hate speech: number of CF users who judged the tweet as hate speech.
3. Offensive language: number of CF users who judged the tweet as offensive language.

4. Neither: number of CF users who judged the tweet as neither.

5. Class: class label for most CF users(0 for hate speech, 1 for offensive language and 2 for neither)

6. Tweet: text to be used for the classification.

2.2 Explorative Data Analysis

Exploratory Data Analysis (EDA) is an approach to analyzing and summarizing datasets in order to gain a better understanding of the data, identify patterns and relationships, and determine any necessary data pre-processing steps. EDA is a crucial step in the machine learning and deep learning project process, as the insights obtained from EDA can guide the selection of appropriate algorithms and features and can improve the model performance. In particular analyzing our data we observe that The dataset is a matrix of the following dimensions 24783 rows $\times$ 7 columns. In particular, being a hate detection we have carried out an analysis of our labels, we have therefore observed that they are unbalanced as 77.4/100 contain label 1 (offessinve Language), 16.8/100 contain label 2 (neither) and the remainder are labels 0 (hate speech). We chose not to apply any changes as we believe it is a good data split for training.

2.3 Input Peprocessing

Our input is a textual input and so must be transformed into something processable and usable as input of a DNN (Deep Neural Network). The process of going from words or sentences to vectors of real numbers is called word embedding. In the following sections we will explain the different embedding techniques used and we will classify them, showing their effectiveness, merits and drawbacks. But before is fundamental to apply some preprocessing. Like any algorithm also word embedding algorithms require the input text to have particular characteristics. In this section are explicate the operations that the input text (tweets) underwent before being used as input to the different possible embedding algorithms. In particular, the tweet has in most input data the following structure '@username: text' and for this reason it is preprocessed by a function defined by us called valid words that applies the following operations:

- Tokenization: a fundamental step in natural language processing that involves breaking down a sequence of text into smaller units, known as tokens. These tokens can be individual words, subwords, or even characters, depending on the granularity of the tokenization process. However, in most NLP applications, words are commonly used as tokens due to their semantic significance in human language. We made the choice of using words as tokens in our project since is primarily motivated by the assumption that words carry important linguistic and semantic information. By breaking text into words, we can analyze and understand the meaning of sentences and documents more effectively. We have made this

choice also based on some papers that demonstrating the Usefulness of Word Tokenization in text classification but we will analyze it later.

- Lower casing: another common preprocessing step in NLP that involves converting all characters in a given text to lowercase. This transformation ensures uniformity and consistency in the textual data and plays a crucial role in various NLP applications. By applying lower case, we treat uppercase and lowercase letters as the same, thereby reducing the dimensionality of the vocabulary and facilitating better generalization. Lower casing in particular helps with the normalization of our tweets making it easier to capture word frequencies and patterns.

- Stemming: Lastly we apply this technique that reduces words to their base or root form, known as the stem. The stem represents the core meaning of a word and is obtained by removing prefixes, suffixes, and inflections. The goal of stemming is to normalize words and reduce them to a common form, which can improve information retrieval, text analysis, and other NLP tasks. Generally another fundamental and similar technique called lemmatization is used, the aim of this preprocessing strategy is to reduce words to their base or canonical form, known as the lemma. The lemma represents the dictionary form or the root form of a word, which carries its fundamental meaning. Unlike stemming, which applies heuristic rules to truncate words, lemmatization takes into account the morphological analysis of words and considers factors such as the part of speech and the context in which the word appears.We have choose stemming and not lemmatization since stemming offers several advantages over lemmatization in certain contexts, in particular stemming algorithms are generally simpler and computationally more efficient compared to lemmatization. Stemming involves rule-based or heuristic approaches that rely on pattern matching and string manipulation, making it faster and easier to implement. Another reason is that Stemming retains the original word shape by preserving the stem's internal structure, which can be beneficial in tasks where the morphology or word shape plays a role. This can be particularly important in information retrieval systems or text mining applications. In particular we have use one of the most important stemming algorithm known as Porter stemming algorithm, developed by Martin Porter in 1980. The Porter stemming algorithm follows a set of predefined rules or heuristics to reduce words to their stems. The algorithm employs a series of transformation steps to remove common suffixes and variations, ultimately aiming to find the root or base form of a word. To implemented it in our code we have used NLTK (Natural Language Toolkit) library, a popular Python library for NLP.

3 Methodology

3.1 Embedding Procedures

3.1.1 What is text embedding ? What we expect from it?

As we have say in the previous section a textual input must be transformed into something processable and usable as input of a DNN (Deep Neural Network). The process of going from words or sentences to vectors of real numbers is called word embedding. In this section we will then look in detail at what should be expected from text embedding in order for it to be considered valid, and to do so we will individually analyze the most famous and popular embedding techniques in the history of natural language processing. The simple idea of embedding strategy is that from a set of words , called vocabulary, it give as output a numerical bijectional representation of our word that has the ability to recognize similarity between different terms. More deeply we could that our embedded representation reach this 3 features:

1. The embedding must unify superficially different words.
2. The embedding must capture information about how words cand be used.
3. The embedding must separate polysemous words in base of the meaning.

3.1.2 Word types embedding

The first techniques we implemented and tested are known as word type embedding techniques. The concept of word type refers to the choice of representing words as unique forms in a text. It focuses on treating each distinct word as a single entity, regardless of its specific occurrences or frequency in the text. These are a set of techniques that initiated and then enabled the growth of NLP. The limitation of these techniques is that even the best of them fail to meet requirement number 3 ("The embedding must separate polysemous words in base of the meaning") and so they have been replaced by a set of techniques called word token embedding based on the use of neural language models which we will discuss in the next section. The word types embedding techniques which share the distributional model intuition could be classified in 3 different main categories:

1. Word Type sparse distributional model: This type of models are based the simple one since represent the distributional properties of word types in a sparse manner. It typically relies on sparse vectors or matrices to represent the distributional information of words. In this model, each word type is associated with a high-dimensional vector sparse representation, where the dimensions correspond to different features or contexts. The model captures co-occurrence patterns of words across a large corpus of text to derive the distributional information. In general the most known are: the co-occurences matrix rapresentation, the PPMI matrix rapresentation, the

sintax based vector rapresentation , the tf-idf matrix rapresentation and the cosine similarity matrix rapresentation. In our project we choose to use the the TF-IDF one.

- The TF-IDF (Term Frequency-Inverse Document Frequency) method was first introduced in the field of information retrieval, but it is also widely used in NLP. The TF-IDF formula is attributed to Karen Spärck Jones, a pioneer in the field of information retrieval, who described the concept in her paper "A Statistical Interpretation of Term Specificity and Its Application in Retrieval" in 1972. The TF-IDF calculates the weight of a term in a document based on its frequency in the document (TF) and its inverse frequency in the entire corpus (IDF). Specifically, it is the product of the following two values:

$$\text{TF-IDF}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{1}$$

  where

$$\text{tf}(t, d) = \frac{\text{number of occurrences of term } t \text{ in document } d}{\text{total number of terms in document } d}$$
$$\text{idf}(t, D) = \log\left(\frac{\text{total number of documents in the corpus } D}{\text{number of documents containing term } t}\right)$$

  To implement it we have TfidfVectorizer from the scikit-learn library to calculate the TF-IDF (Term Frequency-Inverse Document Frequency) characteristics of a text. The TfidfVectorizer class of scikit-learn is an extension of the CountVectorizer vectorizer, which converts a collection of texts into a numerical representation using word frequencies.

2. Word Type dense distributional model: A dense distributional model of word types represents the distributional properties of word types in a dense manner. Unlike the sparse model, it uses dense vectors or embeddings to encode the distributional information. Each word type is represented by a low-dimensional vector in a continuous vector space, where similar words are close to each other. Dense distributional models can be divided into two subcategories: The count base models such as singular value decomposition combined with latent semantic analysis that exploit matrix factorization techniques. The second type are the embedding learning alghorithms models such as word2vec , GloVe or FastText, capture semantic relationships between words by exploiting neural network architectures. Specifically, we have selected Word2Vec as the algorithm to use in the project, this because is less complex than FastText and we are not so interesting in the global information that GloVe could acquire since each tweet is a very small sequence of words and so we decide to use the local information that Word2Vec has the ability to compute.

- In particular Word2Vec is a machine learning model introduced by Tomas Milkov and his research team at Google in 2013 with the paper "Efficient Estimation of Word Representations in Vector Space". Word2Vec is based on the idea that the meaning of a word can be captured by analyzing the contexts in which it appears. The model uses a shallow feedforward neural network to learn dense vector representations of words from the context in which they occur. In particular, the representations are the weights are obtained from the model of the trained net. There are two main variations of the Word2Vec model: Continuous Bag-of-Words (CBOW) and Skip-gram. The first takes multiple words as input the context (a set of words) and returns a single word as output. The second instead does the opposite and gets the context from a single word. As can be easily deduced, we are interested in using the CBOW variance in our project. Continuous Bag-of-Words (CBOW): In this variation, the model tries to predict a target word based on words in the surrounding context. A context of input words is provided and the goal is to predict the target word. CBOW is particularly useful when the text corpus contains shorter sentences and the focus is on predicting the target word based on the surrounding context. The neural network weights represent word embeddings, which are learned by optimizing a loss function. Once trained, the Word2Vec model provides dense vector representations of words, where similar words are represented by closest vectors in vector space. Graphically :

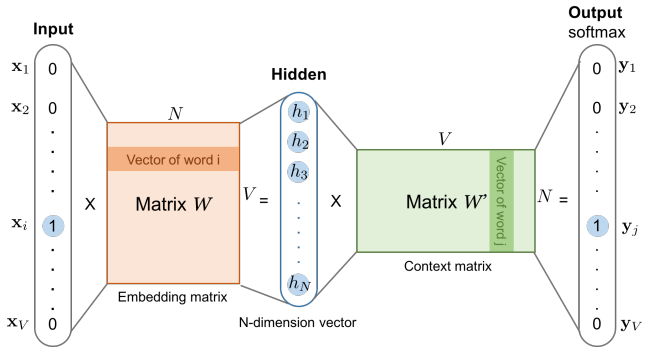


Figure 1: Word2Vec, CBOW variant

3. Word Type sense distributional model: A distributive word type sense model takes into account the different senses or meanings of a word type when modeling its distributive properties. Although models such as SenseEmbed (2015) by Iacobacci or S2W2 by Mancini (2017) could lead to improvements in the result, we choose not to use them as possible models since distributive word-type sense models often require more complex and resource-intensive techniques than simpler models. Another problem in building accurate word-type sense distribution models requires sense-

annotated data, where each occurrence of a word is labeled with the intended sense and this is not the case of our dataset.

3.1.3 Word types embedding

As we have said the word types embedding has a big limitation since has not the ability to separate polysemous words in base of the meaning. For this reason, a new set of techniques was introduced and became a new standard for the NLP words. This set of techniques are based on a different use of the words, in particular not words as types but as tokens (instances of words types) , this means that words embedding algorithm of this kind construct embedding on the fly, so based not just on the word type but also on its context. For this reason, this embedding algorithm are called word contextualized embedding strategy. This kind of embedding is strictly connected with the neural language models, in particular the embedding is obtain as the interpolation of the weights of the neural network used to make the prediction of the next words. In particular in base of the type of network used we could have different kind of neural models and consequently different kind of embedding, the principal one is:

- ELMo : (Embedding from Language Models) was introduced by Peters in 2018 with the paper "Deep contextualized Word Vector Representation" . In particular the embedding is constructed as the interpolation of the state of a bidirectional longh short time memory (a particular kind of recurrent neural network introduce in the 1995 by Sepp Hochreiter) that receives as input a word type embedding as Word2Vec. Graphycally the network could be represented in the following way:

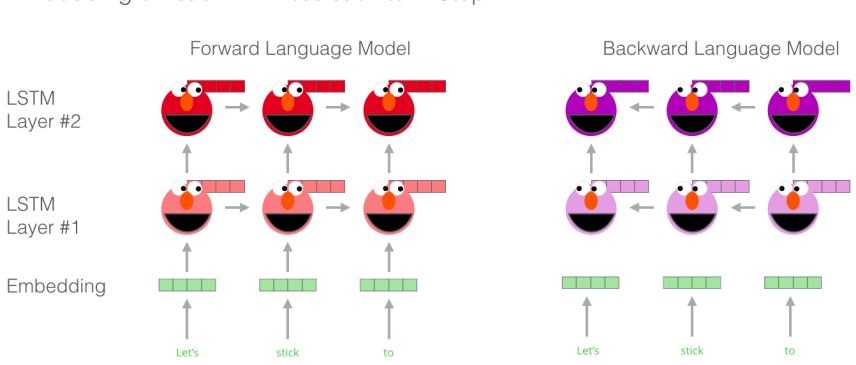


Figure 2: ELMo strcucture

To find the embedding the model use the state obtained from the training and then apply an interpolation formula. More precisely the loss function used in the training is a cross entropy loss and the interpolation formula is :

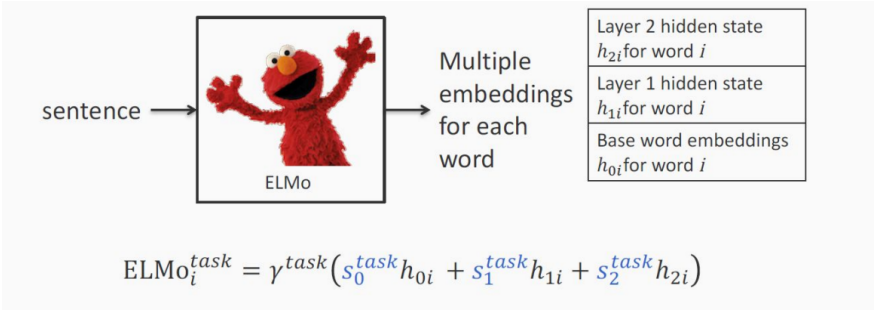


Figure 3: ELMo interpolation to obtain the embedings

3.2 Supervised Single Label Multiclass Classification

Having seen and analyzed the different embedding techniques, it is now crucial to use the different texts generated by the different embeddings in our task, which as mentioned initially is a sentiment analysis, that is, a supervised multiclass classification with a single label. In particular, the construction of our network was done thanks to the Keras python library. Keras is a high-level interface for more specialized, well-optimized tensor manipulation libraries. It serves as the frontend because it handles the model definition, specifying the structure, layers, connections between neurons, and training parameters.

3.2.1 Type of network: MLP and RNN

To accomplish this task we have selected two types of neural network based on the representation. In particular:

- For the sparse embedding (TF-Idfwe utilize a simple deep learning model known as the Multilayer Perceptron (MLP). The MLP is a type of fully connected feedforward deep neural network (DNN). In our project, the MLP comprises three layers: an input layer, a hidden layer, and an output layer. Each node, except for the input nodes, represents a neuron that employs a nonlinear activation function. The MLP leverages a supervised learning technique called backpropagation, which utilizes the chain rule, to train and seek the optimal or suboptimal solution.

- For the dense representation instead we use a simple recurrent neural network compose by a single LSTM, the same used in the construction of the ELMo language model.

3.2.2 Input of the networks: MLP and RNN

For the input layer of our MLP, we will use the different 2D tensors obtained as representations through the TF-IDF embedding algorithms. The input layer will consist of 64 neurons, each of which will have the size of the embedding used as the input size.

For the recurrent we use as a unique layer before the output one a LSTM which receives as input a vector (so a words) for each of the 32 time step. Specifically we have choose 32 since it's the maximum length of our tweets.

3.2.3 Output of the networks: MLP and RNN

For the output layer we have the same for both the networks, we utilized a fully connected layer consisting of n classes, where n represents the number of possible classes (in this case, 3 for each target). Each neuron in this layer utilizes the softmax activation function, which is a generalization of the sigmoid function. The softmax function provides probabilities for each possible class as the output, with one neuron dedicated to each class. The formula used for softmax activation is as follows:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

3.2.4 Hidden layers of the networks: MLP and RNN

As mentioned with the LSTM we have just input and output layer, with the mlp we employ a single hidden layer with 32 neurons, which is half the size of our input layer. In this case, since all of our layers are fully connected, each neuron in the hidden layer receives as input a linear combination of the outputs from the input layer.

3.2.5 Activation function of the networks: MLP AND RNN

The activation function is a mathematical function linear combination of the input of the layer.The purpose of an activation function is to introduce non-linear transformations to the network, enabling it to model and approximate non-linear relationships between inputs and outputs. Without an activation function, the neural network would only be able to learn linear relationships, severely limiting its expressive power. As activation we just choose the one of the MLP since in the LSTM is already chosed. In particular the activation function for our MLP is the ReLU (Rectified Linear Unit). ReLU is one of the most widely used activation functions in the world. Specifically, the ReLU is half (bottom) rectified. f(z) is zero when z is less than zero and f(z) is equal to z when z is greater than or equal to zero. Similar to linear units, but produces zero over half of its domain. In particular the formula is:

$$f(x) = \max(0, x)$$

3.2.6 Loss function of the networks: MLP and RNN

A loss function, also known as an objective function or a cost function, is a mathematical function that quantifies the discrepancy or error between the predicted outputs of a model and the true values of the target variable. It measures

how well the model is performing and provides a signal for the model to adjust its parameters during the training process. In our case, we utilize the sparse categorical cross-entropy as the chosen loss function for both the type of networks. This is a variation of the categorical cross-entropy loss function, specifically designed for situations where the output labels are represented in a sparse matrix format. In a sparse matrix format, the labels are encoded as a single index value rather than a one-hot encoded vector. The sparse categorical cross-entropy measures the difference between the predicted probability distribution (P(x)) and the true distribution (Q(x)) on the same input random variable. It quantifies how well the predicted distribution matches the true distribution, providing a measure of the model's performance. By minimizing the sparse categorical cross-entropy loss, the model aims to optimize its predictions and improve its accuracy in handling the sparse label.

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

3.2.7 Weight Regularizers: MLP and RNN

A regularizer is a technique used in neural networks to prevent overfitting, which occurs when a model becomes too specialized to the training data and performs poorly on new, unseen data. Regularization helps improve the generalization capability of a deep learning model when faced with completely new data from the problem domain. In particular, we will use L2 regularization. L2 regularization, also known as weight decay, is a common regularization technique in which the magnitude of the weights in the model is penalized. This is achieved by adding a regularization term to the loss function, which encourages the weights to be as small as possible. The L2 regularization term is computed by taking the sum of the squares of all the weights in the network and multiplying it by a regularization factor, typically denoted as lambda or alpha. This additional term in the loss function helps prevent the model from relying too heavily on any particular feature or input, promoting a more balanced and generalized representation. By incorporating L2 regularization into the training process, the model learns to strike a balance between fitting the training data well and avoiding overfitting. This regularization technique encourages simpler and smoother weight configurations, leading to improved generalization performance on unseen data.

$$R(W) = \frac{\lambda}{2} \sum_{i=1}^{N} \|W_i\|^2$$

3.2.8 Optimizers: MLP and RNN

An optimizer is an algorithm or method that is used to adjust the weights and learning parameters of a neural network during the training process. It determines how the network learns and updates its parameters based on the loss

function and the gradients computed during backpropagation. In our project we have used Adam, which stands for Adaptive Moment Estimation. Adam combines the advantages of two other popular optimizers, RMSprop and Adagrad, and provides efficient gradient-based updates to the network weights. Adam maintains an adaptive learning rate for each parameter, allowing it to dynamically adjust the learning rate based on the magnitude of the gradients and the past update history. It keeps a running average of both the gradients and the squared gradients of the parameters, which are then used to compute the updates. This adaptive learning rate scheme makes Adam well-suited for dealing with sparse gradients and noisy data.

3.2.9 Batch Size: MLP and RNN

The batch size refers to the number of training examples processed in a single forward and backward pass during the training of a neural network. It is a hyperparameter that determines the trade-off between computational efficiency and the quality of the model's updates. In our project we have used a t technique for setting the batch size called mini-batch training. In mini-batch training, the training dataset is divided into small batches, and the model's parameters are updated based on the average gradient computed from each batch. This approach lies between two extremes: batch training, where the entire dataset is used for each update, and stochastic training, where a single example is used for each update. Mini-batch training offers several advantages over the other two approaches. Firstly, it provides a balance between computational efficiency and model accuracy. Using larger batch sizes allows for better utilization of hardware resources (e.g., GPUs), as parallelism can be leveraged to process multiple examples simultaneously. At the same time, mini-batches provide enough variability to estimate the gradient accurately and update the model parameters effectively.

4 Result

In this section we focus on analyzing two main points. First we analyze the different embeddings obtained in an intrinsic way. As second instead we do an extrinsic analysis of the embeddings by observing how they perform in the hate classfication task by analyzing their strengths, weaknesses and possible improvements.

4.1 Intrinsic Evaluation of the embeddings

In this section we carry out an intrinsic evaluation of the embeddings used. In particular here we focus on the direct evaluation of the quality of the embeddings without involving another model or task. In particular, to do this, we decide to observe the similarity of some pairs of words for each embedding and to create, through a dimensionality reduction, the vector space containing all the words for each strategy used.

In particular, to carry out the similarity analysis we use the cosine similarity. The cosine similarity is a measure of similarity between two vectors in a vector space, is often used in natural language processing tasks to compare the similarity of word embeddings or document vectors. In particular it calculates the cosine of the angle between the two vectors, representing the direction and similarity of their orientations. The formula to calculate cosine similarity between two vectors, v1 and v2, is as follows:

$$\text{cosine similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Choosing the word dumb we obtain in the most similar words ['stupid'] so we could consider the emebedding able to capture a sort of similarity. Instead, to visualize the different word embeddings in a 3-dimensional space, it is possible to use a dimensionality reduction called Principal Component Analysis (PCA). Principal Component Analysis (PCA) is a widely used algorithm for dimensionality reduction in data. The main goal of PCA is to transform a set of related variables into a new set of uncorrelated variables called principal components. In the context of word embeddings, PCA can be applied to reduce the size of embedding spaces, which usually have many dimensions, in a three-dimensional space. This allows word embeddings to be displayed in a three-dimensional graph, making it easier to understand the relationships between words. To apply PCA to word embeddings, we take the set of embedding vectors as the input data. Each word in the vocabulary is represented as a fixed-size numeric array in the word embedding space. These embedding vectors can be organized into a matrix, with each row corresponding to an embedding vector. Next, the PCA algorithm calculates the principal components of the embedding matrix. Once the principal components are obtained, it is possible to reduce the dimensionality of the embeddings by projecting the original vectors onto the subspace generated by the first three principal components. This subspace is a three-dimensional space that represents a compressed version of the original embeddings. In particular for example used the PCA on the CBOW embedding we obtain:

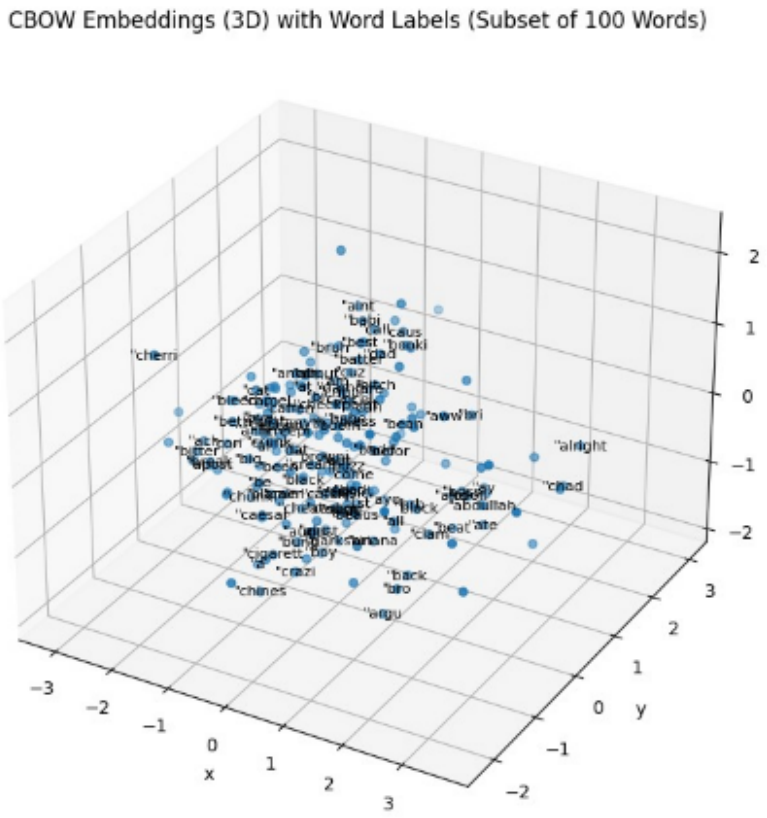


Figure 4: CBOW embedding word space

4.2 Hate Detection results : Extrinsic Evaluation of the embeddings

After having analyzed the different types of embedding in an intrinsic way, in this section we analyze the results obtained in the hate detection task carried out through the MLP described in the previous section. Doing this also corresponds to carrying out an extrinsic analysis of the embedding algorithms. In particular, to analyze the results obtained from the hate detection task we use:

1. Accuracy values on train and test in each model
2. Confusion matrices
3. The Learning Curves

4.2.1 Accuracy Tables

Accuracy is a common measure used to evaluate the performance of a machine learning model in classification. Indicates the percentage of correct predictions out of the total predictions made. In particular, the values are contained in the following table:

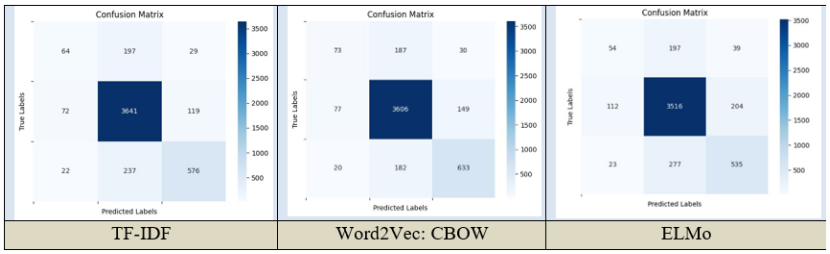
Embedding Model + DNN	Training Accuracy	Test Accuracy
TF-IDF + MLP	0.989	0.854
CBOW (Word2Vec) + LSTM	0.921	0.863
ELMo +LSTM	0.978	0.838

Figure 5: Accuracy value on train and test for each model

It can be seen from this table that the model has more accurate values on train data than on test data. To visualize the motivations better we also perform the analysis by confusion matrix and learning curves.

4.2.2 Confusion Matrix

The confusion matrix, also known as an error matrix, is a fundamental tool for evaluating the performance of a multiclass classification model. It provides valuable insights into the model's predictions by showing the number of correct and incorrect predictions made for each output class. The confusion matrix is organized in a table format, where the rows represent the true classes and the columns represent the predicted classes. The elements within the matrix indicate the number of instances that have been correctly or incorrectly classified for each class pair. This information is extremely useful as it provides a detailed overview of the model's performance for each output class. From the confusion matrix, various evaluation metrics can be calculated, such as accuracy, precision, recall. Our confusion matrix is :



TF-IDF | Word2Vec: CBOW | ELMo

Figure 6: Confusion Matrix of the three model.

Generally a good confusion matrix should exhibit a dominant diagonal with higher values along the main diagonal. This indicates that the model made correct predictions for the majority of instances in each class. Additionally, it is desirable to have low values off the main diagonal, as they represent classification errors. Ideally, all cells off the main diagonal to have low values. In our case

although not identical the confusion matrices show some small imprcision in agreement with the result obtained on the accuracy

4.2.3 Learning Curve

The learning curve provides insights into the model's behavior based on the size of the training set. It consists of two curves: the training accuracy and the validation accuracy. From different models we obtain the following learning curves :

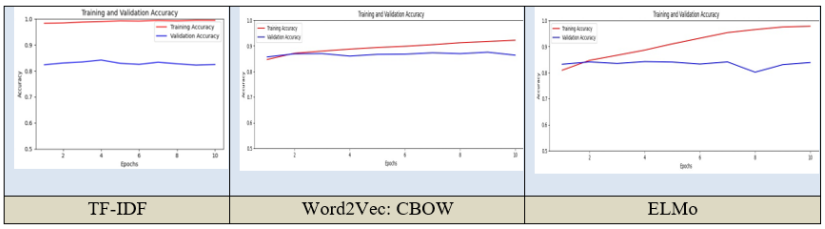


Figure 7: Learning curve of the three model

A good learning curve shows a progressive increase in training accuracy as the size of the training set grows. This indicates that the model is learning from the training data and becoming better at correctly predicting labels. The validation accuracy should initially increase with the growth of the training set size. This indicates that the model is able to generalize well from the training data to the test data. However, after a certain point, the validation accuracy may stabilize or even decrease, which can be caused by factors such as model complexity or noise in the data. A convergence point between the training and validation curves indicates that the model has reached a balance between underfitting and overfitting. If the curves are very close or overlap, it suggests that the model is effectively learning from the training data and generalizing well to the test data. Combining what we have seen here with what we have seen in the previous two sections, we then observe that all three models can be considered valid but with some inaccuracies. In fact, even the newer models have problems on generalization and that is why there remains some overfitting on all three models. Despite this they still have a good ability to generalize having a predictive ability of about 8 out of 10 comments.

5 Conclusion

5.1 Summary of the project

In conclusion, this project has showcased the potential of artificial intelligence, specifically natural language processing, in addressing the issue of hate speech detection. Throughout the study, we followed a systematic approach that encompassed several key points.

First, we delved into the introduction of hate speech detection, highlighting the significance of this problem and its potential consequences. We then explored the role of artificial intelligence and natural language processing in tackling this issue.

The data used for this project underwent explorative data analysis and input processing, ensuring a comprehensive understanding of the dataset. We employed various embedding procedures, which allowed us to represent the text in a meaningful way and set the expectations for these embedding techniques.

Furthermore, we examined different types of word embeddings and their effectiveness in hate speech detection. The supervised single-label multiclass classification approach was adopted, employing both multilayer perceptron (MLP) and recurrent neural network (RNN) models. We discussed the input, output, hidden layers, activation functions, loss functions, regularizers, optimizers, and batch sizes used in these networks.

The results of the project were evaluated through intrinsic and extrinsic methods. Intrinsic evaluation focused on assessing the quality and performance of the embeddings. Extrinsic evaluation involved hate detection results, which were measured using accuracy tables, confusion matrices, and learning curves.

Overall, the findings of this study demonstrate that although hate speech remains a significant concern, the integration of artificial intelligence and natural language processing offers potential solutions. Through the development and implementation of effective models, we can detect and address instances of hate speech, contributing to a safer and more inclusive online environment.

5.2 Shared and personal roles

The project was carried out and designed by Giacomo Colosio and Mattia Moro. Specifically, the conception of the content was done jointly, then given the impossibility of working in the same place led us to have Mattia be more involved in the implementation part of the project while Giacomo was more in the theoretical explanations and report creation. Despite this, however, everything was done by producing together and confronting each other on every occasion where needed

Refereces

- Presentation of the course of Text Mining and natural Language Processing 2022-2023 BSc AI University of Pavia
- Presentation of the course of Machine Learning and deep convolutional neural Netowrk 2022-2023 BSc AI University of Pavia
- "Speech and Language Processing" by Daniel Jurafsky and James H. Martin. Available at: `https://web.stanford.edu/~jurafsky/slp3/6.pdf`
- Wikipedia: `https://www.wikipedia.org/`
- GitHub: `https://github.com/`
- Keras: `https://keras.io/`
- Scikit-learn: `https://scikit-learn.org/`
- NLTK (Natural Language Toolkit): `https://www.nltk.org/`
- Deep Learning Book: `http://www.deeplearningbook.org/`
- Coursera Deep Learning Specialization: `https://www.coursera.org/specializations/deep-learning`
- arXiv: `https://arxiv.org/`
- IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI): `https://www.computer.org/csdl/journal/tp`