



UNIVERSITY OF TRIESTE

Department of Mathematics, Informatics and Geosciences

Bachelor's Degree in Artificial Intelligence and Data Analytics

BACHELOR'S THESIS

**When Words Hurt: Machine Learning Recognition of Misogyny
Patterns in Social Networks**

Supervisor:

Prof. Alejandro Garcia Rodriguez

Candidate:

Margherita
Necchi

Academic Year 2024/2025

Abstract

The growing proliferation of offensive content on social media represents a significant challenge, within which misogyny, as a form of hate speech directed at women because they are women, has become an increasingly relevant issue. The need to develop automatic systems for identifying such content is crucial to promoting a safer and more respectful online environment.

The goal of this thesis is the design, implementation and comparative evaluation of different Machine Learning and Deep Learning methodologies for the task of automatic misogyny recognition in Italian texts. The work was carried out on the tweet dataset provided for the Automatic Misogyny Identification (AMI) task of the EVALITA 2020 conference.

The methodological approach involves a systematic comparison between two families of models. First, classical classification models were implemented and optimized, including Naive Bayes, Logistic Regression and Support Vector Machines (SVM). Subsequently, a Deep Learning architecture based on Convolutional Neural Networks (CNN) was explored. For the latter, a pre-trained embedding specific to the social media domain was employed in order to effectively capture the semantic relationships in the text.

The results show that, although well-optimized classical models constitute a robust and highly competitive baseline, the CNN architecture achieves superior performance, demonstrating greater effectiveness in learning the hierarchical and complex features of hate speech.

This study compares the effectiveness of different generations of text classification models, providing a practical analysis of strategies for identifying misogyny in Italian and confirming the value of neural architectures for this task of great social relevance.

Contents

Abstract	i
Contents	iii
List of Figures	v
List of Tables	vii
1 Introduction	1
2 Theoretical background	3
2.1 Word Embedding	3
2.1.1 GloVe	3
2.2 Linear Models	3
2.2.1 Logistic Regression	3
2.2.2 Support Vector Machines (SVM) with Linear Kernel	4
2.3 Probabilistic Models	5
2.3.1 Naive Bayes (Multinomial)	5
2.4 Tree-based Models	5
2.4.1 Decision Trees	5
2.4.2 Random Forest	6
2.5 Neural Networks	6
2.5.1 Convolutional Neural Networks (CNN)	6
2.6 Evaluation Metrics for Classification	7
3 Dataset and Preprocessing	9
3.1 Dataset Description	9
3.2 Exploratory Analysis	9
3.2.1 General Statistics and Class Distribution	9
3.2.2 Tweet Length Analysis	10
3.3 Preprocessing Pipeline	11
3.3.1 Data Cleaning	11
3.3.2 Normalization	12

3.3.3	Tokenization and Lemmatization	12
3.3.4	"Disambiguation"	12
3.4	Post-preprocessing Dataset Analysis	13
3.4.1	New Statistics	13
4	Experimental Methodology	17
4.1	Feature Representation	17
4.1.1	TF-IDF	17
4.1.2	Word Embedding	17
4.2	Configuration	18
4.2.1	Traditional Models	18
4.2.2	CNN	20
5	Results	23
5.1	Results of the Classical Models	23
5.2	Results of the CNN	25
5.3	Comparative Analysis	26
	Bibliography	31
	Declaration	33

List of Figures

2.1	Example taken from Soni et al. 2022 [15].	7
3.1	Word cloud of the most frequent words in the text, split by Class 0 (left) and Class 1 (right).	10
3.2	Distribution and proportion of classes in the original dataset.	10
3.3	Distribution of the length (characters and words) of the original text, split by class.	11
3.4	Comparison of the length distribution (words and characters) between the original text (blue) and the processed text (orange).	13
3.5	Comparison of the 20 most frequent words before and after preprocessing. . .	14
4.1	Class distribution in the Training, Validation and Test sets. Note how the class proportions are maintained in each split.	21
5.1	Comparison of the ROC curves of the classical classification models.	23
5.2	Loss and Accuracy during the training of the optimized CNN.	25
5.3	Graphical analysis of the performance of the optimized CNN model on the test set.	26

List of Tables

3.1	Examples of the application of the preprocessing pipeline.	15
4.1	Hyperparameter grid for the grid search	19
5.1	Performance of the classical classification models on the test set. The values reported are the F1-score (macro) and the AUC (Area Under the ROC Curve).	23
5.2	Optimal hyperparameter configuration for the classical models, identified through Grid Search.	24
5.3	Comparison of the final performance of the best classical model and the CNN.	26
5.4	Top 12 most important features for the random forest.	27

Chapter 1

Introduction

The advent of the digital era and the pervasiveness of social networks have radically transformed the ways in which people communicate and interact socially. Platforms such as X, Facebook and Instagram have become the new global public squares, offering unprecedented opportunities for sharing information and for public debate. However, this digital revolution also has a dark side: the exponential spread of hate speech, disinformation and verbal violence.

Within the vast landscape of hate speech, online misogyny is one of the most harmful and pervasive manifestations. It is not a matter of simple insults, but of a system of aggression, harassment and denigration directed against women, which aims to belittle them, intimidate them and exclude them from public debate. This phenomenon has severe real-world consequences: it normalizes gender-based violence, reinforces harmful stereotypes and can have a devastating psychological impact on its victims. Countering online misogyny is therefore not only a technical problem, but a pressing social need.

This thesis is set in this context. Its main objective is to explore, implement and critically compare different Machine Learning and Deep Learning methodologies for the task of automatically recognizing misogyny in Italian texts extracted from the social network X. Leveraging Natural Language Processing (NLP) techniques, this thesis aims to answer a fundamental question: how effectively, and with what limitations, can computational models understand the complex nuances of hate speech directed at women and assist in its moderation? To carry out this analysis, the reference dataset of the AMI (Automatic Misogyny Identification) task of the EVALITA 2020 conference was used.

The choice of this topic stems not only from academic interest, but also from a deep awareness of its social relevance. The fight against misogyny and gender-based violence is one of the most urgent challenges of our time, a path that requires the commitment of every discipline. This thesis, within the limits of its scope, is therefore meant to be a contribution, however small, to this collective effort, applying the tools of machine learning to a problem of fundamental importance for a better society. The ambition and the hope are that every step forward in technology may in some way also be a step towards the end of gender inequality.

Chapter 2

Theoretical background

2.1 Word Embedding

In order to feed texts to deep learning models such as Convolutional Neural Networks (CNN), words must be converted into a numerical representation. A basic approach such as bag of words, which represents texts by counting the frequency of each word, has two main drawbacks: a huge dimensionality and the inability to capture relationships of meaning and word order.

To overcome these limitations, word embedding is used: a technique that represents words in a continuous numerical space through dense, low-dimensional vectors. The advantage is twofold. First, dimensionality is drastically reduced: for example, with a vocabulary of 10,000 words, a bag of words vector would have 10,000 dimensions, whereas a word embedding typically has 100 or 300 dimensions, making computations much more efficient. Second, and more importantly, this technique captures semantic and syntactic relationships between words: terms with similar meanings will have nearby vectors in the embedding space, allowing the model to understand context more effectively.

2.1.1 GloVe

For this thesis, pre-trained word embeddings were chosen in order to exploit the linguistic knowledge already learned from huge amounts of text. In particular, the GloVe (Global Vectors for Word Representation) model [13], developed by Pennington, Socher and Manning, was adopted. A version of GloVe trained specifically on billions of tweets was selected: a choice that ensures that the vector representation of words is more faithful to the informal language, rich in slang and neologisms, that is typical of the social network under analysis.

2.2 Linear Models

2.2.1 Logistic Regression

Logistic Regression is a binary classification model whose purpose is to predict a discrete output variable. The model uses the logistic function to compute the probability that a given observation belongs to one of the two classes.

The core mechanism of the model consists in mapping the output of a linear function,

$z = w^T x + b$, into a probability interval $[0, 1]$ through a squashing function. The function used for this purpose is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (2.1)$$

The output, $p(x) = \sigma(w^T x + b)$, is interpreted as the probability of belonging to the positive class. The final classification is obtained by applying a threshold α (typically 0.5) to this probability. The model parameters, namely the weights w and the bias b , are learned by minimizing a cost function. For logistic regression, the cross-entropy loss function is used. The formula is the following:

$$\mathcal{L}(w, b) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))] \quad (2.2)$$

where y_i is the true label (0 or 1) and $p(x_i)$ is the probability predicted by the model for the i -th observation.

A fundamental property of this cost function, in the context of logistic regression, is its convexity with respect to the parameters w and b . Although the cross-entropy function is not convex in general, its application to a linear model followed by a sigmoid function (as in logistic regression) produces a convex cost surface. This guarantees that optimization algorithms, such as gradient descent, can converge to a single global minimum, thus reliably finding the optimal parameters of the model.

2.2.2 Support Vector Machines (SVM) with Linear Kernel

Support Vector Machines (SVM) are a class of supervised learning models whose goal is to identify the optimal hyperplane separating points belonging to two distinct classes. The guiding principle of an SVM is the search for the maximum-margin hyperplane, that is, the one positioned at the greatest possible distance from the nearest points of each class.

Mathematically, this translates into a constrained optimization problem, in which the norm of the weight vector is minimized (equivalent to maximizing the margin), subject to the constraint that all points are correctly classified:

$$\min_{w, b} \|w\|_2 = \min_{w, b} w^T w \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 0, \quad \forall i \quad (2.3)$$

where w and b define the hyperplane and (x_i, y_i) are the points of the dataset with their labels $y_i \in \{-1, 1\}$.

To solve this constrained problem, the method of Lagrange multipliers is used. This technique transforms the original problem into its "dual" form, introducing a multiplier α_i for each point of the dataset. The solution of the dual problem has a fundamental property: the multipliers α_i are non-zero only for the points lying exactly on the margin. These points are the support vectors, and they are the only ones that determine the position of the hyperplane, hence the name of the model.

The model adopted in this thesis is a Support Vector Machine with a Linear Kernel. This approach is particularly well suited to text classification tasks. Indeed, the high dimensionality that characterizes textual data once vectorized often implies that the classes are already linearly separable, without the need to resort to the non-linear transformations provided by other types of kernels.

2.3 Probabilistic Models

2.3.1 Naive Bayes (Multinomial)

The Naive Bayes classifier is a probabilistic supervised learning model whose functioning is based on the application of Bayes' Theorem. The goal of the model is to compute the probability that a given sample, described by a feature vector x , belongs to a certain class c . Mathematically, this is expressed as:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \quad (2.4)$$

where $P(c|x)$ is the posterior probability, $P(x|c)$ is the likelihood, $P(c)$ is the prior probability of the class and $P(x)$ is the evidence.

The fundamental and distinctive characteristic of this algorithm is the naive assumption that all features $x_1, x_2, \dots, x_n$ are conditionally independent of each other, given the class c . This assumption greatly simplifies the computation of the likelihood $P(x|c)$, which becomes the product of the probabilities of the individual features:

$$P(x|c) = P(x_1|c)P(x_2|c) \dots P(x_n|c) = \prod_{i=1}^n P(x_i|c) \quad (2.5)$$

In the context of text classification, this means that the model assumes that the presence of a word in a sentence is an event independent of the presence of any other word. Although this is a strong simplification of linguistic reality, it allows the model to be very computationally efficient.

There are several variants of the model, each suited to a different type of data. For text classification tasks, where the features represent word frequencies, the most appropriate variant is Multinomial Naive Bayes, which is the one used in this study. The model computes the posterior probability for each possible class and chooses the one that maximizes it. The decision rule, known as maximum a posteriori (MAP) estimation, is therefore:

$$\hat{y} = \arg \max_c P(c) \prod_{i=1}^n P(x_i|c) \quad (2.6)$$

Note that the denominator $P(x)$ is omitted, since it is constant for all classes and does not affect the choice of the class with the highest probability.

2.4 Tree-based Models

2.4.1 Decision Trees

Decision Trees are supervised learning models that are highly valued for their high interpretability. Their hierarchical structure makes it possible to represent a set of decisions and their possible consequences in an easily understandable format, similar to a flowchart. In a decision tree, each internal node represents a test on an attribute (feature), each branch represents a possible outcome of the test, and each leaf node corresponds to a class label.

A tree is typically built through a recursive top-down process, starting from a root node that contains the entire dataset. At each step, the algorithm must select the best attribute on which to perform a split. The selection criterion is based on the ability of an attribute

to create subsets of data that are as pure as possible, i.e., containing samples that belong predominantly to a single class.

Several metrics are used to measure the purity of a node. One of the most common is entropy, which measures the degree of disorder or uncertainty of a dataset S . For a classification problem with k classes, entropy is defined as:

$$E(S) = \sum_{i=1}^k -p_i \log_2(p_i) \quad (2.7)$$

where p_i is the proportion of samples belonging to class i in node S . The algorithm then chooses the attribute that maximizes the information gain, that is, the reduction in entropy obtained after performing the split.

A tree that is allowed to grow without constraints tends to become excessively complex and to memorize the noise present in the training data, a phenomenon known as overfitting. To avoid this, pruning techniques are used.

2.4.2 Random Forest

Random Forest is a supervised learning model that belongs to the category of ensemble learning methods. Instead of relying on a single model, a Random Forest builds a multitude of decision trees during the training phase and combines their predictions to obtain a more accurate and stable final result. The main goal of this technique is to reduce variance and, consequently, the risk of overfitting to which a single decision tree is subject.

The strength and robustness of the model derive from the introduction of randomness in the process of building the forest.

2.5 Neural Networks

2.5.1 Convolutional Neural Networks (CNN)

Although originally conceived for image analysis, Convolutional Neural Networks (CNN) have also been adapted with remarkable success to Natural Language Processing tasks, including text classification [15]. The key intuition is to treat the sequence of word embeddings of a text as a one-dimensional matrix. Filters (or kernels) operate on this matrix, sliding along the sequence to identify local patterns, effectively acting as detectors of meaningful n-grams or phrases. Through successive layers, the CNN is able to compose these low-level features into more complex semantic representations.

The architecture of a CNN for text classification typically consists of several sequential layers:

1. **Embedding layer:** The textual input is first converted into a sequence of word embeddings. This transforms the text into a numerical matrix $X \in \mathbb{R}^{L \times d}$, where L is the length of the sequence and d is the dimension of the embeddings. This matrix is the actual input for the subsequent layers.
2. **Convolutional layer ('Conv1D'):** This is the heart of the model. In this layer, kernels $w \in \mathbb{R}^{h \times d}$ are applied, where h is the window size (e.g., 3, 4 or 5 words). Each filter slides along the embedding sequence to detect local patterns, acting as an n-gram

detector. The result of applying a filter is a feature map c . Each element c_i of the feature map is computed as:

$$c_i = f(w \cdot x_{i:i+h-1} + b) \quad (2.8)$$

where f is a non-linear activation function (such as ReLU), $x_{i:i+h-1}$ is the embedding sub-matrix on which the filter is operating, and b is a bias term.

3. **Pooling layer ('MaxPooling'):** After the convolution, a pooling layer reduces the dimensionality of the feature maps. A very common technique in NLP is global max pooling, which extracts the single maximum value from an entire feature map $c = [c_1, c_2, \dots, c_m]$. The operation is defined as:

$$\hat{c} = \max\{c_1, c_2, \dots, c_m\} \quad (2.9)$$

This operation provides geometric robustness with respect to deformations relative to a hierarchy of patterns, keeping only the most relevant features.

4. **Fully-Connected and Output layers:** The values extracted from all the filters, $\hat{c}$, are concatenated into a single vector. This vector is finally passed to one or more dense (fully-connected) layers that perform the final classification. For a binary problem, the output is a single neuron with a sigmoid activation function, which computes the final probability p :

$$p = \sigma(W_f \cdot z + b_f) \quad (2.10)$$

where z is the concatenated vector coming from the pooling layer and W_f, b_f are the weights and bias of the output layer.

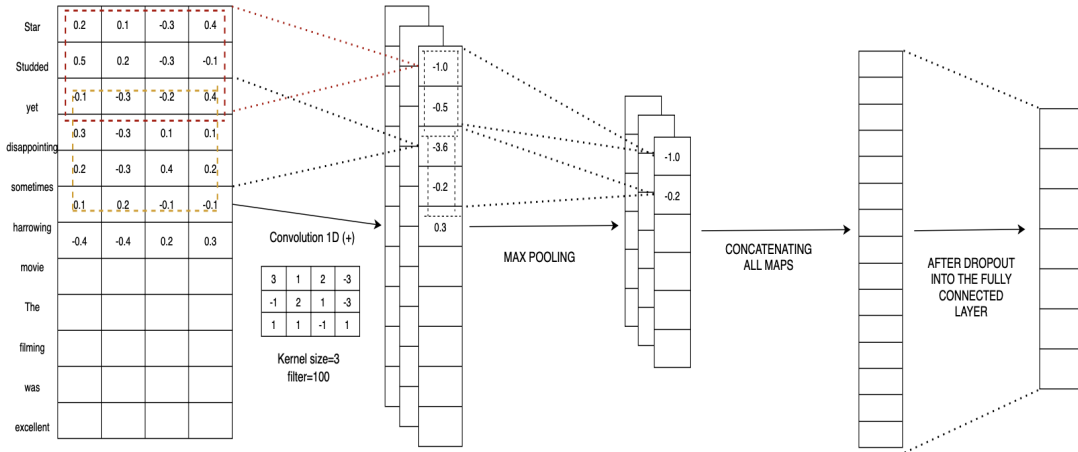


Figure 2.1: Example taken from Soni et al. 2022 [15].

2.6 Evaluation Metrics for Classification

In order to objectively evaluate and compare the performance of the classification models developed, it is essential to use a set of standard quantitative metrics. Each metric offers a different perspective on the classifier's ability to distinguish between classes. The fundamental metrics used in this study are based on the four possible outcomes of a binary prediction:

- **True Positives (TP):** Positive-class samples correctly classified.

- **True Negatives (TN):** Negative-class samples correctly classified.
- **False Positives (FP):** Negative-class samples incorrectly classified as positive (Type I Error).
- **False Negatives (FN):** Positive-class samples incorrectly classified as negative (Type II Error).

These values are typically summarized in a confusion matrix and form the basis for computing the following evaluation metrics:

1. **Accuracy:** The fraction of correct predictions out of the total number of predictions made. It is the most intuitive metric, but it can be misleading with imbalanced datasets, where a model could achieve high accuracy simply by always predicting the majority class. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.11)$$

2. **Precision:** Measures the proportion of positive predictions that were actually correct. It answers the question: "Of all the samples that the model identified as misogynistic, how many really were?". High precision indicates a low number of false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.12)$$

3. **Recall:** Measures the proportion of actual positive samples that were correctly identified. It answers the question: "Of all the truly misogynistic tweets, how many did the model find?". High recall indicates a low number of false negatives.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.13)$$

4. **F1-Score:** The harmonic mean of precision and recall, providing a single value that balances both metrics. It is particularly useful in settings with imbalanced classes, since a high F1-Score requires both precision and recall to be high. For this reason, it was chosen as the reference metric for the optimization and evaluation of the models in this thesis.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \quad (2.14)$$

Chapter 3

Dataset and Preprocessing

3.1 Dataset Description

The dataset used in this thesis, as already mentioned, was provided for the Automatic Misogyny Identification (AMI) task of the EVALITA 2020 conference [4]. It is a collection of Italian-language texts extracted from the social network X (formerly Twitter), which were manually labeled for the specific task of recognizing misogynistic language.

The dataset consists of 13,115 Italian tweets, which represent a heterogeneous sample of informal language, often characterized by slang expressions, typos and neologisms typical of the platform.

As specified in the task guidelines, each instance of the dataset consists of the text of the tweet and a manual label defining its nature: misogynistic, aggressive, both or neither.

In order to focus the analysis on the primary task of misogyny recognition, the classification problem was simplified with respect to the original formulation of the task. Specifically, only the binary label concerning the presence or absence of misogynistic content was considered, excluding from the analysis the secondary distinction regarding the level of aggressiveness. This made it possible to focus the evaluation of the models on their fundamental ability to discriminate gender-based hate speech.

3.2 Exploratory Analysis

Before applying any preprocessing technique, an exploratory data analysis was carried out on the original dataset to understand its fundamental characteristics.

3.2.1 General Statistics and Class Distribution

The preliminary analysis of the dataset reveals a slight imbalance in the class distribution, as illustrated in Figure 3.2. Out of a total of 13,115 tweets, 7,653 (58.4%) are labeled as non-misogynistic (Class 0), while the remaining 5,462 (41.6%) are classified as misogynistic (Class 1). Although the imbalance is not extreme, it suggests the importance of using robust evaluation metrics, such as the F1-score, which are less sensitive to class distribution than simple accuracy.



Figure 3.1: Word cloud of the most frequent words in the text, split by Class 0 (left) and Class 1 (right).

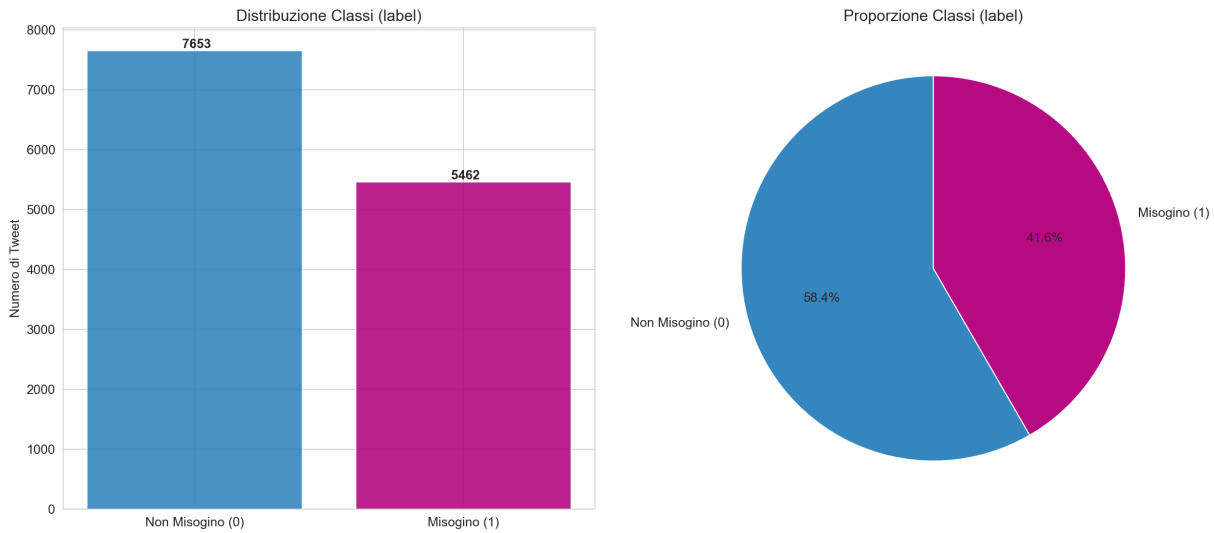


Figure 3.2: Distribution and proportion of classes in the original dataset.

3.2.2 Tweet Length Analysis

The distribution of character length shows a bimodal tendency, with one peak for short tweets (under 100 characters) and a second, more moderate peak close to the maximum character limit then allowed by X (280 characters). The distribution of the number of words is strongly skewed, with most tweets concentrated between 5 and 25 words. In both plots, it can be seen that misogynistic tweets (Class 1) tend to be shorter on average, probably suggesting a more concise and direct language.

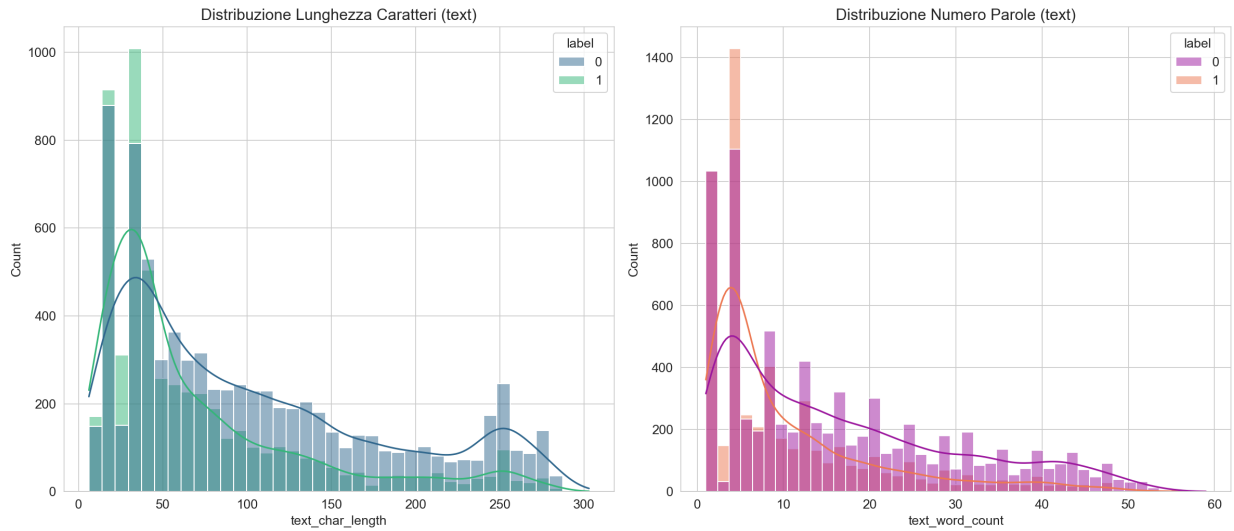


Figure 3.3: Distribution of the length (characters and words) of the original text, split by class.

3.3 Preprocessing Pipeline

Text coming from social networks, and from X in particular, is generally characterized by a high level of "noise": it contains slang, abbreviations, typos, emojis and platform-specific structural elements such as URLs and mentions. To make this data usable by machine learning models and to reduce the complexity of the vocabulary, a multi-stage preprocessing pipeline was applied. The goal is to normalize and clean the text while preserving the relevant semantic information.

3.3.1 Data Cleaning

The first stage focused on removing or replacing elements that do not carry a direct lexical meaning but can introduce noise.

- **Removal of URLs, mentions and hashtags:** URLs, references to other users (mentions, e.g. @user) and hashtags (#) were handled to simplify the text. URLs were completely removed, while mentions were replaced with a generic placeholder, `mention`. This removes the variability of usernames, which do not directly carry meaning, while keeping the information that the tweet was a direct interaction. Hashtags were stripped of the # symbol, but the word they contain was kept in the text, since it is often an integral part of the message.
- **Emoji handling:** emojis and emoticons, while conveying a strong emotional and sentiment signal, greatly increase the complexity of the vocabulary. For this reason, they were replaced with a single placeholder, `emoji`, so as to mark their presence without having to handle hundreds of different symbols.
- **Removal of special characters and punctuation:** punctuation and special characters (e.g. *, &, \$) were removed. Special treatment was reserved for punctuation marks that can indicate emotional emphasis, such as exclamation and question marks, whose presence was encoded with a dedicated placeholder.

3.3.2 Normalization

The second stage focused on standardizing the remaining text in order to group together different forms of the same word or concept.

- **Lowercasing:** all text was converted to lowercase. This step is essential to ensure that words such as "Donna", "donna" and "DONNA" (Italian for "woman") are treated as a single token, avoiding needlessly inflating the vocabulary.
- **Abbreviations and slang:** social media language is full of abbreviations and slang (e.g. `xkè` for "perché", i.e. "why/because"). For this purpose, a dictionary of common and widespread abbreviations was defined and then used to expand these forms into their standard version, improving the consistency of the text.
- **Elongated words:** users often use elongated words to better convey emotions (e.g. `ciaooo` for "ciao"). These colloquial forms were normalized by reducing sequences of repeated letters to a maximum of two, bringing the term back to its correct form.

3.3.3 Tokenization and Lemmatization

Once cleaned and normalized, the text was split into atomic units, or tokens, which typically correspond to single words. Finally, a lemmatization process was applied. Unlike stemming, which simply cuts off the end of a word, lemmatization reduces each word to its base dictionary form (the "lemma"), for example transforming "andando", "andai" and "andrei" (inflected forms of the verb "to go") into "andare". This step, performed with the spaCy library for the Italian language, is crucial for drastically reducing the dimensionality of the vocabulary and for grouping all the inflections of a verb or the declensions of a noun under a single concept.

3.3.4 "Disambiguation"

One of the most complex challenges in automatic misogyny recognition lies in the semantic ambiguity of language. Misogynistic intent, in fact, is often conveyed not by intrinsically offensive words, but by the use of neutral terms in a pejorative context. For example, a word like "cagna" (Italian for "bitch") can refer to a female dog in a neutral context, or be used as a serious insult in a hateful context. Traditional models, which rely on word frequency or static embeddings, struggle to capture this contextual difference.

To address this problem, an additional preprocessing stage was implemented, aimed at disambiguating the meaning of a predefined set of ambiguous terms, inspired by recent research work in the field [[muti-et al-2024-pejorativity](#)]. The goal of this step is to enrich the text by replacing ambiguous words with a substitute (or "anchor") that makes their contextual meaning explicit, in order to provide a clearer and less ambiguous input to the subsequent classification models.

The process is based on three key components:

1. **Lexicon:** first of all, a custom lexicon was defined that maps a series of ambiguous words to a pair of textual "anchors". For each word, one anchor representing its pejorative sense and one representing its neutral sense were defined. For example, for the word "cagna", the anchors defined were "insulto donna" ("insult woman", pejorative) and "cane femmina" ("female dog", neutral).

2. **Classification model:** to determine which of the two senses was appropriate in a given tweet, a pre-trained classification model (`MoritzLaurer/mDeBERTa-v3-base-mnli-xnli`) was employed, accessible through the Transformers library. For each ambiguous word found in a tweet, the model receives as input the entire text of the tweet and the two anchors (pejorative and neutral) as candidate labels. The classifier then determines which of the two labels/anchors is more appropriate to describe the context of the tweet, returning the most likely sense.
3. **Replacement:** once the contextual sense of the word has been determined between the two provided, the original word in the tweet is replaced with the corresponding textual anchor. For reasons of computational efficiency, the disambiguation and replacement process was applied only to the first ambiguous word found in each tweet.

3.4 Post-preprocessing Dataset Analysis

After applying the entire cleaning and enrichment pipeline, a second analysis phase was carried out to evaluate the impact of these transformations on the dataset and to get an overall view of the data that will be used.

3.4.1 New Statistics

The most evident effect of the preprocessing pipeline is the significant reduction and standardization of text length. Figure 3.4 compares the distribution of tweet length (in words and characters) before and after the cleaning process.

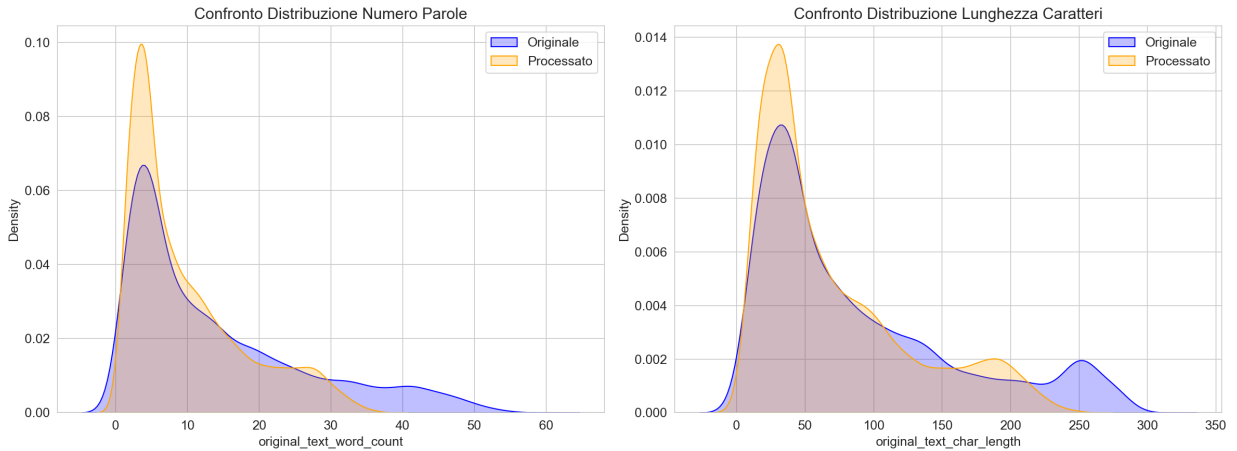


Figure 3.4: Comparison of the length distribution (words and characters) between the original text (blue) and the processed text (orange).

As can be observed, the distribution of the processed text (in orange) is clearly shifted to the left, with a sharper peak at lower length values. This indicates that the removal of "noisy" elements such as URLs, mentions, punctuation and stopwords successfully reduced the length of the texts, eliminating the tails of very long tweets and making the input format more uniform. The distribution, originally bimodal, now takes on a more unimodal shape.

To objectively quantify the characteristics of the corpus and the impact of the preprocessing pipeline, a statistical analysis based on term frequency was carried out. For this analysis, all tokens from the training partition of the dataset, before and after preprocessing, were aggregated to build two distinct frequency maps.

A fundamental part of the analysis concerned Out-of-Vocabulary (OOV) words, i.e., terms not present in a reference dictionary. The vocabulary of the **AIBERTO** language model, specific to the Italian language [14], was used as reference. The set of unique words of each corpus (original and processed) was compared with this vocabulary to identify the OOV terms. To measure their impact, two percentage metrics were computed: the percentage over *types* (unique words) and the percentage over *tokens* (total occurrences).

Applying this methodology produced the following results. The vocabulary of the original text, consisting of 20,007 unique words, is reduced to **13,342** after preprocessing, a decrease of about 33%, which testifies to the effectiveness of lemmatization and normalization.

The OOV analysis shows an even clearer improvement: the percentage of unique out-of-vocabulary words drops from 52.2% in the original text to 33.4% in the processed text. The impact of these words on the text as a whole is even more significant: their total frequency (tokens) goes from 18.3% to only 10.8%. This confirms that the cleaning pipeline successfully removed or corrected a large number of rare terms, probably typos or slang, making the final vocabulary cleaner and more standard.

Finally, Figure 3.5 shows a visual comparison of the 20 most frequent words before and after preprocessing, highlighting how terms with a stronger and potentially more discriminating meaning for the classification task emerge from the processed text.

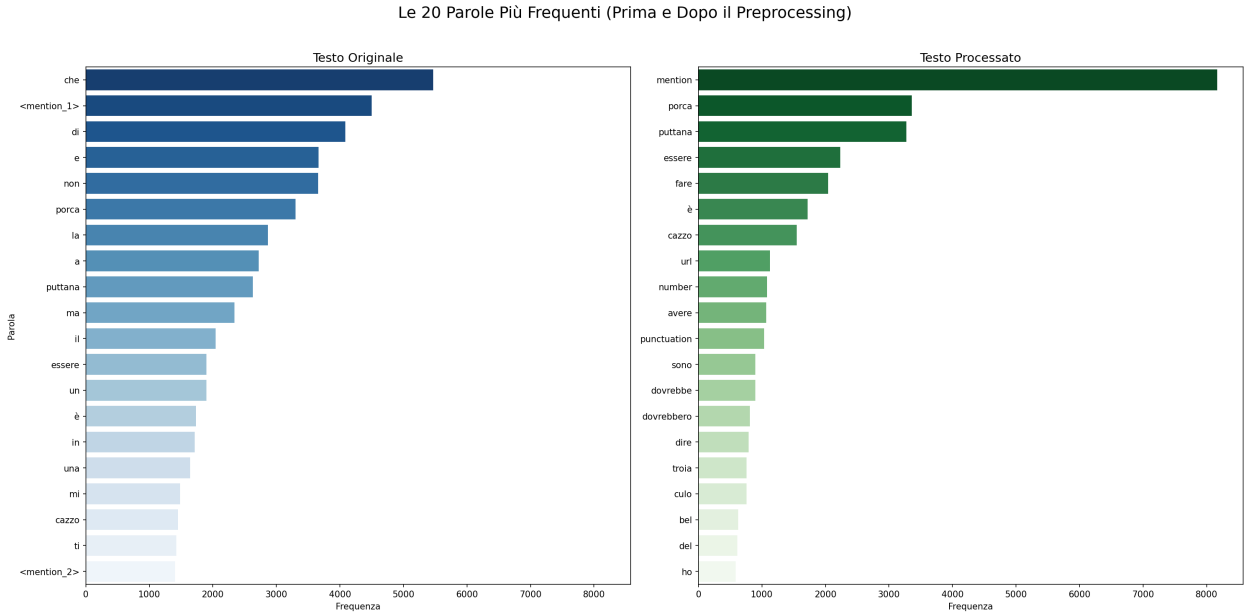


Figure 3.5: Comparison of the 20 most frequent words before and after preprocessing.

Finally, to illustrate the overall impact of the preprocessing pipeline, two example tweets are shown below, one misogynistic and one non-misogynistic, before and after the transformations applied. Table 3.1 summarizes the key steps.

Table 3.1: Examples of the application of the preprocessing pipeline.

Original Tweet	Processed Tweet	Main Transformations Applied
@Ejramruj ma statte zitta ke non capisci nnt! sei solo una cagna da sottomettere duramente	<mention_d> stare zitto capire niente essere solo insulto donna sottomettere duramente	• Normalization (lowercase, slang "ke", "nnt"). • Replacement of mentions with a tag. • Lemmatization ("statte" → "stare"). • Stopword removal (e.g. "ma", "le", "come", ...).
CCronache dall'ascensore.Porca puttana sono bloccata aaaahh https://t.co/k74o9RH5C5	cronaca dall ascensore porca puttana essere bloccare ahh <url>	• Lowercasing. • Replacement of URLs with a tag. • Lemmatization ("bloc-cata" → "bloccare").

Chapter 4

Experimental Methodology

4.1 Feature Representation

In order to be processed by a machine learning algorithm, a text must be transformed into a numerical vector representation. Two distinct strategies were adopted in this study: an approach based on term frequency for the classical models and one based on semantic embeddings for the neural architecture.

4.1.1 TF-IDF

For the classical machine learning models (Naive Bayes, Logistic Regression, SVM, etc.), the **TF-IDF (Term Frequency-Inverse Document Frequency)** technique was used. This method represents each tweet as a numerical vector in which each dimension corresponds to a word in the vocabulary. The value of each dimension is not a simple count, but a weight computed by combining two factors:

- **Term Frequency (TF):** Measures how often a term appears within a single tweet. A term that is more frequent in a document will have a higher TF value.
- **Inverse Document Frequency (IDF):** Measures the importance of a term on a global scale (the entire dataset). Terms that are very common across all documents (e.g. articles, prepositions) are assigned a low IDF weight, while rarer and potentially more informative terms are assigned a higher weight.

The final TF-IDF score for a word in a document is the product of these two values. This approach has the advantage of giving greater prominence to words that are frequent in a specific text but rare in the rest of the dataset, making them highly discriminating features for classification.

4.1.2 Word Embedding

For the deep learning architecture (CNN), a semantically richer representation based on pre-trained **word embeddings** was chosen. As described in the previous chapter, these models represent words as dense vectors in a multidimensional space where distance and direction capture semantic relationships.

As already mentioned, for this thesis it was chosen to use pre-trained **GloVe** vectors.

The specific configuration used is the following:

- **Model:** GloVe (glove.twitter.27B).
- **Vector Dimension:** 100 dimensions.

During the preprocessing phase, each tweet was converted into a sequence of numerical indices, where each index corresponds to a word in the vocabulary. These indices were then used to retrieve the corresponding 100-dimensional vectors from the GloVe matrix, thus generating the input matrix for the Convolutional Neural Network.

4.2 Configuration

The architectures and specific configurations of all the implemented models are now described. For each model, the optimal configuration was sought through a hyperparameter tuning process.

4.2.1 Traditional Models

For the classification phase with classical approaches, several machine learning models were implemented and evaluated, including **Multinomial Naive Bayes**, **Logistic Regression**, **Support Vector Machine (SVM) with Linear Kernel**, **Decision Tree** and **Random Forest**.

All these models were trained using as input the TF-IDF feature vectors described in the previous section. To identify the optimal hyperparameter configuration for each model, a Grid Search strategy with 5-fold cross validation was used. This process, as mentioned earlier, was guided by the maximization of the F1-score.

The grid with the hyperparameters tested for each model is reported below.

Table 4.1: Hyperparameter grid for the grid search

Model	Pipeline Parameter	Tested Values
MultinomialNB	<code>sel__k</code> (No. of features)	[500, 600, 700]
	<code>tfidf__use_idf</code> (Use IDF)	True, False
	<code>clf__alpha</code> (Smoothing)	1.0, 0.1, 0.01
LogisticRegression	<code>sel__k</code> (No. of features)	[600, 700, 800]
	<code>tfidf__norm</code> (Normalization)	'l1', 'l2'
	<code>clf__C</code> (Inv. regularization)	<code>logspace(-2, 2, 5)</code>
	<code>clf__penalty</code> (Penalty type)	['l1', 'l2']
LinearSVC	<code>sel__k</code> (No. of features)	[600, 700, 800]
	<code>tfidf__use_idf</code> (Use IDF)	True, False
	<code>clf__C</code> (Inv. regularization)	<code>logspace(-2, 2, 5)</code>
DecisionTree	<code>sel__k</code> (No. of features)	[400, 500, 600, 700]
	<code>clf__criterion</code> (Split criterion)	['gini', 'entropy']
	<code>clf__max_depth</code> (Max depth)	[None, 10, 20, 30]
	<code>clf__min_samples_split</code>	[2, 10, 20]
	<code>clf__min_samples_leaf</code>	[1, 5, 10]
RandomForest	<code>sel__k</code> (No. of features)	[400, 500, 600]
	<code>clf__n_estimators</code> (No. of trees)	[10, 20, 25]
	<code>clf__criterion</code> (Split criterion)	['gini', 'entropy']
	<code>clf__max_depth</code> (Max depth)	[None, 10, 25]
	<code>clf__min_samples_split</code>	[2, 10, 20]
	<code>clf__min_samples_leaf</code>	[1, 5, 10]

C (Inverse of regularization) Used in logistic regression and SVM, this parameter controls the penalty associated with classification errors. A lower value of C leads to a simpler model with a wider separation margin, while a high value pushes the model to correctly classify more training examples, at the risk of overfitting.

penalty Specific to logistic regression, it determines the type of norm used for regularization (L1 or L2). L1 implies Lasso regularization, L2 Ridge.

k (SelectKBest) Refers to the number of features (in this case, the words with the highest TF-IDF scores) to select before feeding them to the classifier. It serves to reduce dimensionality and noise, keeping only the most informative terms. The different values depend on the robustness of the model to overfitting and noise.

alpha (Smoothing) Used in Multinomial Naive Bayes, it is the smoothing parameter. It

serves to handle words not present in the training set, preventing the model from assigning them a probability of zero.

criterion Used in Decision Tree and Random Forest, it is the function used to measure the quality of a split. The options tested are the Gini impurity index and entropy, both methods for evaluating how "pure" a node is (i.e., whether it contains samples of a single class).

max_depth The maximum depth allowed for a tree. It is a crucial hyperparameter for controlling the complexity of Decision Tree and Random Forest and preventing overfitting.

min_samples_split The minimum number of samples required to split an internal node in a decision tree. This parameter also serves to control the growth of the tree.

n_estimators Specific to Random Forest, it represents the total number of decision trees to build within the "forest". A larger number of trees generally leads to a more robust model, at the cost of a higher computational cost.

4.2.2 CNN

For the deep learning approach, a **Convolutional Neural Network (CNN)** was designed and implemented. It was chosen to keep the model architecture as simple as possible to avoid the risk of overfitting. Each type of layer is therefore present only once. The final architecture is the following:

1. **Embedding layer:** The input layer receives fixed-length sequences (35 tokens). This layer uses the pre-trained **GloVe** embedding matrix (100-dimensional vectors) to convert tokens into dense vectors. This layer was made trainable, allowing the model to fine-tune the GloVe vectors to adapt them specifically to the context of misogyny in Italian social media language.
2. **Convolutional layer ('Conv1D'):** A one-dimensional convolutional layer with 64 filters and a kernel size of 3. This layer is responsible for extracting local patterns (n-grams) from the embedding sequence, using the ReLU activation function.
3. **Pooling layer ('GlobalMaxPooling1D'):** A Global Max Pooling layer is applied to the output of the convolutional layer. In this case, global pooling was chosen, as it is generally useful for producing an invariant representation.
4. **Regularization layer ('Dropout'):** To counter overfitting, a dropout layer with a rate of 50% was added, which randomly deactivates half of the neurons during each training cycle.
5. **Output layer:** The final layer is a single dense neuron with a sigmoid activation function, which outputs a probability between 0 and 1, suitable for binary classification.

To train the network, the Adam optimizer with a learning rate of 0.001 and the binary cross-entropy loss function, specific to binary classification problems, were used.

To ensure a robust and unbiased evaluation of the models, the complete dataset was split into three disjoint sets:

- **Training Set (60% of the data):** Used exclusively for training the models.
- **Validation Set (20% of the data):** Used for hyperparameter tuning and for intermediate decisions during training (such as early stopping for the CNN).

- **Test Set (20% of the data):** A completely untouched set, used only once at the end of the process for the final evaluation of the performance of the optimized model.

The split was performed in a stratified manner to ensure that the class proportions were maintained in all three subsets, as shown in Figure 4.1.

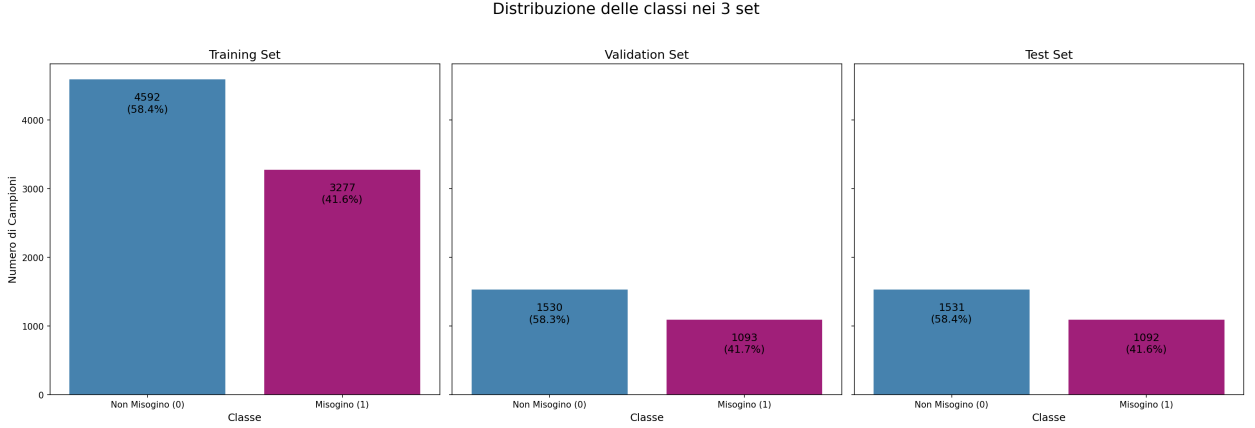


Figure 4.1: Class distribution in the Training, Validation and Test sets. Note how the class proportions are maintained in each split.

The validation strategies were then adapted to the two families of models:

- For the classical models: the hyperparameter search (grid search) was performed by applying a 5-fold cross validation strategy on the training set only. The validation set was not used in this phase. Once the model with the best configuration had been identified, its final performance was evaluated on the test set.
- For the CNN: the model was trained on the training set, while its performance was monitored epoch by epoch on the validation set to drive the Early Stopping technique. Finally, the final model with the best weights was evaluated on the test set to obtain the definitive performance metrics.

This three-way approach ensures that the final evaluation is performed on data that the model has never encountered at any stage of its training or optimization, providing an honest estimate of its generalization ability.

Furthermore, for training the CNN, the early stopping technique was employed to prevent overfitting and to identify the optimal moment to stop training. The process was monitored by tracking the loss function computed on the validation set.

A patience of 5 epochs was set: if the validation loss showed no improvement (i.e., no decrease) for five consecutive epochs, training was automatically stopped. In addition, the option to restore the model weights to the state of the best epoch (the one with the lowest validation loss) was enabled, ensuring that the final saved model was the one with the best generalization ability on unseen data.

Chapter 5

Results

This final chapter presents and analyzes the results obtained by the different classification models implemented. The main metric for the comparison is the (macro) F1-score, alongside the AUC score to evaluate the overall discriminative ability.

5.1 Results of the Classical Models

The traditional machine learning models presented earlier were trained using TF-IDF features and optimized through Grid Search. Table 5.1 summarizes the final performance obtained by each classifier on the test set.

Table 5.1: Performance of the classical classification models on the test set. The values reported are the F1-score (macro) and the AUC (Area Under the ROC Curve).

Model	F1-Score (macro)	AUC
Random Forest	0.8291	0.9328
Logistic Regression	0.8046	0.9087
Support Vector Machine	0.8041	0.9065
Multinomial Naive Bayes	0.7917	0.8912
Decision Tree	0.8242	0.8946

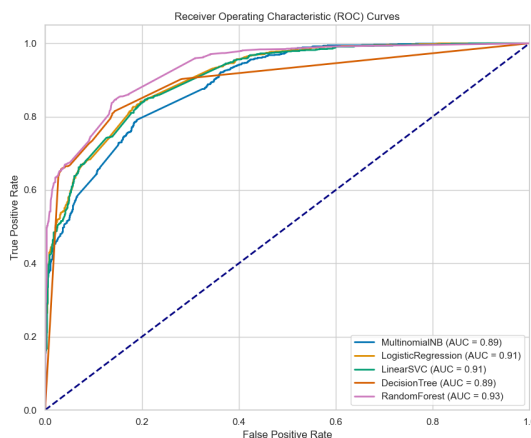


Figure 5.1: Comparison of the ROC curves of the classical classification models.

The visual comparison of performance is illustrated in Figure 5.1, which shows the ROC curves of all the models. The random forest curve lies clearly above the others, confirming the superiority outlined by the previous metrics.

The optimal parameters identified for each model, with which the results reported above were obtained, are listed in Table 5.2.

Table 5.2: Optimal hyperparameter configuration for the classical models, identified through Grid Search.

Model	Hyperparameter	Optimal Value
Random Forest	n_estimators (new)	20
	max_depth (new)	25
	n_estimators (old)	50
	k (No. of features)	600
Decision Tree	criterion	entropy
	max_depth	None
	min_samples_split	2
	min_samples_leaf	1
	k (No. of features)	700
Logistic Regression	C (Regularization)	10.0
	penalty	l2
	k (No. of features)	800
	Tfidf (norm)	l2
Support Vector Machine	C (Regularization)	1.0
	k (No. of features)	800
	Tfidf (use IDF)	True
Multinomial Naive Bayes	alpha (Smoothing)	0.01
	k (No. of features)	700
	Tfidf (use IDF)	False

5.2 Results of the CNN

The final configuration of the model was reached through an iterative optimization process. Several techniques were tested, including the adoption of a dynamic learning rate, but it was found that a configuration with a lower fixed learning rate ($1e-4$) and a simpler architecture (32 filters) produced the most stable results.

The training progress of the final model is shown in Figure 5.2. The learning curves reveal a model that generalizes well, although a slight but evident sign of overfitting emerges in the final stages. As can be observed, after the first 20 epochs the training and validation curves begin to gradually diverge: the loss on the training set keeps decreasing, while the loss on the validation set flattens out. This indicates that, although the phenomenon was kept under control with various techniques, the model begins to memorize some specificities of the training data.

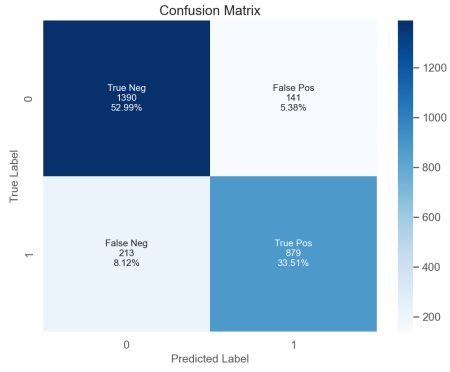
This behavior, however slight, confirms the crucial importance of the early stopping technique. Thanks to it, in fact, the model weights corresponding to the point of minimum validation loss were restored for the final evaluation, thus ensuring that the version of the model with the greatest generalization ability was used, before overfitting became more pronounced.



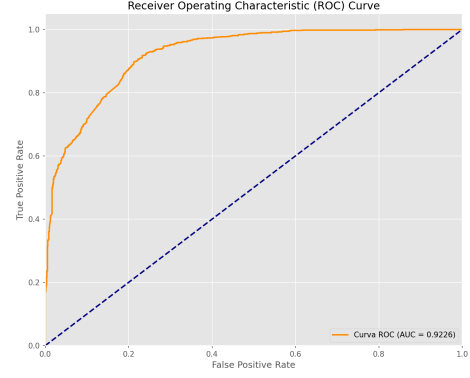
Figure 5.2: Loss and Accuracy during the training of the optimized CNN.

The final evaluation of the model reported a (macro) F1-score of **0.83**, an AUC score of **0.919** and an overall accuracy of **83.6%**.

Performance is analyzed in detail in the figures below. The confusion matrix (Figure 5.3a) shows a good classification ability for both classes. In particular, a high Recall (about 80%) can be observed on the misogynistic class, indicating that the model is very effective at identifying tweets containing gender-based hate speech. The ROC curve (Figure 5.3b) visually confirms the strong discriminative ability of the model, with an area under the curve (AUC) close to 0.92.



(a) Confusion matrix for the CNN model.



(b) ROC curve for the CNN model.

Figure 5.3: Graphical analysis of the performance of the optimized CNN model on the test set.

5.3 Comparative Analysis

To conclude the analysis, this section directly compares the performance of the best classical machine learning model identified, the **random forest**, and the optimized deep learning model, the **convolutional neural network (CNN)**.

Table 5.3: Comparison of the final performance of the best classical model and the CNN.

Model	Precision (macro)	Recall (macro)	F1-score (macro)	AUC
Random Forest	0.841	0.823	0.829	0.933
Convolutional Neural Network (CNN)	0.830	0.830	0.830	0.927

As the table shows, the two models reach an extremely competitive level of performance, with almost equivalent results. The CNN obtains a slightly higher macro F1-score (0.830 vs 0.829), indicating a marginally better balance between precision and recall. Surprisingly, the random forest records a slightly higher AUC value (0.933 vs 0.927), suggesting greater reliability in separating the classes at the probabilistic level.

This closeness in performance shows how a classical model, when supported by a solid preprocessing pipeline and careful feature selection, can compete on equal terms with more complex deep learning architectures. However, a qualitative analysis of their strategies and their classification errors reveals fundamental differences in how they work.

The strength of the random forest lies in its ability to identify and weigh highly predictive key terms. Table 5.4 lists the 12 most relevant features identified by the model.

Table 5.4: Top 12 most important features for the random forest.

Feature (term)	Importance
porca	0.096085
porca puttana	0.091817
mention	0.048370
puttana	0.042934
tettona	0.025447
cazzo	0.024505
culona	0.016195
schiaffi	0.014924
bel	0.013745
sono	0.012837
url	0.010959
dire	0.010116

As can be observed, the model relies heavily on a lexicon of explicit insults (‘porca’, ‘puttana’, ‘cazzo’), terms closely linked to the objectification of the female body (‘tettona’, ‘culona’) and metadata such as mentions and URLs (‘mention’, ‘url’). This strategy makes it an excellent detector of insults and direct offensive patterns.

Its limitation, however, clearly emerges from the error analysis. The model proves fragile in the face of linguistic nuances such as **negation** and **sarcasm**. For example, it wrongly classifies as misogynistic non-offensive tweets that actually condemn violence, since it focuses on a single trigger word while ignoring its context.

Example of a false positive:

“@giornalisteTV @Leone09748724 @Fiero70600786 @Priscil79855548 Beh, non mi sento sottomessa o malmenata” (“Well, I don’t feel subjugated or beaten up”)

Analysis: The model focuses on the word “malmenata” (“beaten up”), associated with a high importance, but it is unable to interpret the negation “non” (“not”), which completely reverses its meaning.

At the same time, it does not recognize misogyny in sentences whose aggressiveness is expressed through sarcasm or with words that, individually, are not vulgar, violent or explicitly misogynistic.

Example of a false negative:

@daechwita00 Ti apro in due (“I’ll split you in two”)

Analysis: The violent syntactic structure is not captured. The model does not find the explicit words or patterns it would expect (such as ‘puttana’ or ‘cazzo’) and wrongly classifies the tweet as non-misogynistic.

This is where the advantage of the CNN, however small, becomes apparent. Its hierarchical nature, capable of analyzing sequences of words through convolutions, allows it to partially overcome the limitations of the classical model. Although not immune to errors, the CNN has a greater ability to:

1. **Adapt to context:** Fine-tuning the embedding layer allows the pre-trained vectors to

specialize in the jargon of the dataset, capturing nuances that a static approach such as TF-IDF might miss.

2. **Understand local syntax:** By analyzing n-grams in sequence, the CNN can learn that the presence of “non” (“not”) before a verb reverses its meaning, reducing false positive errors related to negation.

An alternative and promising approach to overcome the limitations observed would consist in using Large Language Models (LLMs) as extractors of contextual representations. Instead of training a model from scratch, the intermediate representations (hidden states) of models pre-trained on billions of tokens could be exploited to capture even subtler semantic and pragmatic nuances. The process would work by feeding each tweet to a pre-trained LLM and extracting the output tokens of the intermediate layers as dense, highly contextualized features. These representations, enriched by the linguistic knowledge of the pre-trained model, could then be used as input for simpler classifiers (e.g. logistic regression or SVM) or for lightweight neural architectures. The advantages of this approach would indeed be manifold.

In the first place, a deep contextual understanding could be achieved, a key capability in this context: LLMs, trained on huge text corpora, have developed an implicit understanding of complex linguistic phenomena such as sarcasm, irony, negation and conversational implicatures.

There would also be dense and rich representations: unlike TF-IDF, which is based on lexical frequency, LLM representations capture semantic, syntactic and pragmatic relationships in high-dimensional vector spaces.

For the problematic cases identified in this analysis, an LLM would be expected to correctly interpret the negation in sentences such as "non mi sento... malmenata" ("I don't feel... beaten up"), generating representations that reflect the semantic inversion, and to recognize implicit threats in constructions such as the one seen in the example, "Ti apro in due" ("I'll split you in two"), having probably encountered similar patterns during pre-training. Finally, it would be able to distinguish sarcastic contexts from literal ones, thanks to its ability to model long-range dependencies in the text.

This direction would represent a natural evolution of the present work, combining the computational power of LLMs with the simplicity and interpretability of traditional classifiers, potentially offering the best of both approaches.

Acknowledgements

To all the wonderful women in my life.

Bibliography

- [1] F. Bianchi et al. “PejorativITy: A Novel Resource for the Unsupervised Detection of Pejorative Language and Misogyny”. In: *Proceedings of the Language Resources and Evaluation Conference (LREC)*. Marseille, France, June 2022, pp. 6780–6789.
- [2] L. Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.
- [3] C. Cortes and V. Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.
- [4] E. Fersini, D. Nozza, and P. Rosso. “AMI@EVALITA2020: Automatic Misogyny Identification”. In: *Proceedings of the 7th evaluation campaign of Natural Language Processing and Speech tools for Italian (EVALITA 2020)*. CEUR Workshop Proceedings. 2020.
- [5] E. Fersini, D. Nozza, and P. Rosso. *Automatic Misogyny Identification (AMI) Shared Task at EVALITA 2020*. Sito Web Ufficiale. Accesso in data: 19 giugno 2025. 2020. URL: <https://amievalita2020.github.io/>.
- [6] GeeksforGeeks. *Bag of words (BoW) model in NLP*. <https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/>. Accesso: 26-06-2025. 2019.
- [7] M. Honnibal et al. “spaCy: Industrial-strength Natural Language Processing in Python”. In: *Proceedings of the 17th Python in Science Conference*. Ed. by A. Meurer, A. Smith, and K. Class. 2020, pp. 425–434. DOI: 10.25080/Majora-342d178e-035.
- [8] D. Jurafsky and J. H. Martin. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, 2009.
- [9] Y. Kim. “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. URL: <https://www.aclweb.org/anthology/D14-1181>.
- [10] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: <http://tensorflow.org/>.
- [11] E. W. Pamungkas, V. Basile, and V. Patti. “Misogyny detection in Twitter: a multilingual and cross-domain study”. In: *Information Processing & Management* 57.6 (2020), p. 102360.
- [12] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

- [13] J. Pennington, R. Socher, and C. Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: <https://aclanthology.org/D14-1162/>.
- [14] M. Polignano et al. “AlBERTo: Italian BERT-based models for a wide range of NLP tasks”. In: *Proceedings of the Sixth Italian Conference on Computational Linguistics (CLiC-it 2019)*. Bari, Italy: Accademia University Press, Nov. 2019, pp. 1–7. URL: <https://www.aclweb.org/anthology/2019.clic-it-1.21>.
- [15] S. Soni, S. S. Chouhan, and S. S. Rathore. “TextConvoNet: a convolutional neural network based architecture for text classification”. In: *Applied Intelligence* 53.11 (Oct. 2022), pp. 14249–14268. ISSN: 0924-669X. DOI: 10.1007/s10489-022-04221-9. URL: <https://doi.org/10.1007/s10489-022-04221-9>.
- [16] T. Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.

Declaration

The original version of this thesis was written in Italian. This English translation was produced with the assistance of generative artificial intelligence (Claude).