UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

Toxic Comment Classification

Machine learning for MDE project

Revisioni

Revisione	Data	Autori	Descrizione
1.0	13/01/2014	Matteo Cavasinni	document created

Università degli Studi dell'Aquila

- matteo.cavasinni@student.univaq.it

Elenco delle figure

CAPITOLO 1

Introduction

1.1 abstract

1.2 Dataset Description

This dataset was specifically designed for training machine learning models to classify online comments into six distinct categories: toxic, severe toxic, obscene, threat, insult, and identity hate. The dataset comprises a substantial collection of 153,164 unique comments, each labeled with one or more of these categories. Importantly, the dataset is complete, meaning there are no missing values or incomplete entries, ensuring the reliability of the training process.

1.3 Technologies

In this section all the technologies used during the experiment will be listed.

1.3.1 Tensorflow

TensorFlow is an open-source machine learning framework developed by Google Brain. It provides a flexible ecosystem for building, training, and deploying machine learning and deep learning models across various platforms, including CPUs, GPUs, and TPUs. Designed for both researchers and developers, it supports low-level operations for maximum control as well as high-level APIs like Keras for ease of use. TensorFlow enables efficient computation through data flow graphs and automatic differentiation, making it suitable for a wide range of applications, from computer vision to natural language processing and large-scale distributed training.

1.3.2 NLTK

NLTK (Natural Language Toolkit) is a powerful open-source library for natural language processing (NLP) in Python. It provides tools for text processing, tokenization, stemming, lemmatization, part-of-speech tagging, parsing, and semantic reasoning. Designed for both educational and research purposes, NLTK includes access to a vast collection of linguistic datasets and corpora. While it is highly flexible and feature-rich, it is often considered slower than modern alternatives like spaCy. It remains a valuable tool for text analysis, prototyping NLP models, and understanding fundamental NLP concepts.

1.3.3 Spacy

spaCy is a fast and efficient open-source library for natural language processing (NLP) in Python, designed for industrial applications. It provides pre-trained models for tasks like tokenization, part-of-speech tagging, named entity recognition (NER), dependency parsing, and text classification. It supports **large-scale text processing**, with built-in support for deep learning frameworks like TensorFlow and PyTorch. spaCy also offers **customizable pipelines**, allowing users to fine-tune models and add custom components. With its easy-to-use API and multi-language support, spaCy is widely used in AI applications, including chatbots, information extraction, and automated text analysis.

1.3.4 Glove

GloVe (Global Vectors for Word Representation) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.The different versions of GloVe are mainly distinguished by the size of the vocabulary and the embedding vector. Here are the most common versions:

- **glove.6B.zip:** trained on a corpus of 6 billion tokens and includes embedding vectors for about 400,000 words.
- **glove.840B.300d.zip:** trained on a corpus of 840 billion tokens and includes 300-dimensional embedding vectors for about 2.2 million words.
- **glove.twitter.27B.zip:** trained on a corpus of 27 billion tokens from Twitter and includes embedding vectors for about 1.2 million words.

In our case, we will use the 6 billion parameter GloVe model because it is the lightest among them. We will try the various versions (50d, 100d, 200d, 300d) to see the differences between them.

1.3.5 Azure ML

Azure Machine Learning (Azure ML) is a cloud-based platform by Microsoft for building, training, and deploying machine learning models at scale. A key component of Azure ML is Azure ML Notebooks, an interactive environment built on Jupyter notebooks that allows data scientists and developers to write, test, and execute ML code directly in the cloud. Azure ML Notebooks offer integrated compute resources, meaning users can run notebooks on scalable cloud-based virtual machines or clusters, eliminating the need for local setup. These notebooks are tightly integrated with the Azure ML work-space, enabling seamless access to datasets, models, and experiment tracking through the Azure ML SDK for Python. They support collaborative development, with version control via Git and easy sharing across teams. Users can also schedule and automate notebook runs with Azure ML Pipelines, making them a powerful tool for reproducible and scalable machine learning workflows.

CAPITOLO 2
Models and Encoding

2.1 Models

2.1.1 Multi Layer Perceptron

2.1.1.1 Description

The Multilayer Perceptron (MLP) is a type of artificial neural network that processes information in a unidirectional manner, from the input layer to the output layer, without cycles or feedback. Its structure is characterized by the presence of an input layer, one or more hidden layers, and an output layer, each composed of interconnected neurons (or nodes). The input layer is responsible for receiving incoming data, with a number of neurons corresponding to the number of attributes or features of the data itself. The hidden layers, on the other hand, perform intermediate processing on the input data. The network can have one or more hidden layers, and the number of neurons in each layer is a hyperparameter that can be adjusted during the training phase. Finally, the output layer produces the network's final result, with a number of neurons depending on the nature of the problem being addressed (e.g., binary classification, multiclass classification, regression).

2.1.1.2 Architecture

This neural network is designed for multi_label classification, specifically for text analysis and identifying different forms of "toxicity." Let's analyze each layer:

Embedding Layer: This is the first layer and is responsible for transforming the input text (presumably a sequence of words or characters) into a dense vector representation. The embedding_layer is an embedding layer that maps each word/character to a fixed-size vector. The output x is a sequence of these embedding vectors, one for each element of the input. The size of these vectors is a hyperparameter that is defined during the construction of the network. This layer is crucial because it allows the network to understand the semantic meaning of words and their relationships.

Global Average Pooling 1D: This layer applies a Global Average Pooling along the time dimension (the sequence). It takes the average of all embedding vectors for each dimension of the vector. In other words, if the output of the embedding layer had dimension (sequence-length, embedding-dimension), the output of this layer will have dimension (embedding-dimension). This layer reduces dimensionality and provides a global representation vector of the entire input sequence. It is a way to obtain a fixed representation of the sentence regardless of its length.

Dense (128 neurons, ReLU): This is a fully connected layer (Dense) with 128 neurons and ReLU activation function. It receives as input the vector of dimension (embedding_dimension) from the previous layer and transforms it into a vector of dimension 128. ReLU (Rectified Linear Unit) is a non-linear activation function that introduces non-linearity into the model, essential for learning complex patterns. relu(x) returns max(0, x).

Dense (64 neurons, ReLU): Similar to the previous layer, this is another Dense layer with 64 neurons and ReLU activation. It further reduces the dimensionality to 64. Its function is to continue learning more abstract and complex representations of the data.

Dense (32 neurons, ReLU): Again a Dense layer, this time with 32 neurons and ReLU activation. It continues the feature extraction process, reducing the dimensionality to 32.

Output Layer (toxic,severe_toxic,obscene,threat,insult,identity_hate): The other five output layers are structured identically to the toxic_output layer. Each of them produces a single value between 0 and 1, representing the probability that the text belongs to that specific category of ”toxicity.” This structure indicates that the model is designed for multi-label classification, where a single text can belong to multiple categories simultaneously (e.g., a text can be both ”toxic” and ”offensive”).

2.1.2 Convolutional Neural Network

2.1.2.1 Description

Convolutional Neural Networks (CNNs), or convolutional neural networks, are a specialized class of artificial neural networks particularly effective in processing data with a grid-like structure, such as images or videos. Their architecture is inspired by the way the human brain processes visual information.

A CNN is composed of several layers, each with a specific function. The heart of a CNN is the convolutional layer, where the convolution operation takes place. In this layer, a filter (or kernel) slides over the input image, performing a series of multiplications and sums. This process allows the network to detect local patterns or features in the image, such as edges, corners, or textures.

After the convolutional layer, we often find a pooling layer. Pooling aims to reduce the dimensionality of the data and increase invariance to small translations or distortions of the image. There are several pooling techniques, such as max pooling (which selects the maximum value in a region) or average pooling (which calculates the average of the values in a region).

The results of the convolutional and pooling layers are then processed by one or more fully connected layers, similar to those found in a Multilayer Perceptron (MLP).

These layers combine the features extracted from the previous layers to produce the final output of the network, which can be an image classification, an object detection, or something else.

CNNs are distinguished by their ability to learn hierarchies of features. The first convolutional layers detect simple features, such as edges or angles, while the subsequent layers combine these features to recognize more complex patterns, such as shapes or objects. This hierarchical learning ability makes CNNs particularly suitable for computer vision tasks, such as image classification, object detection, facial recognition, or semantic segmentation.

2.1.2.2 Architecture

Embedding Layer: This initial layer converts the input text (words or characters) into dense vector representations called embeddings. It's the starting point for understanding the semantic content.

Conv1D (128 filters, kernel size 3, ReLU): This is a 1dimensional convolutional layer. It applies 128 different filters of size 3 across the embedded text. Think of these filters as feature detectors. ReLU adds nonlinearity. This layer learns patterns in short sequences of words.

MaxPooling1D (pool size 3): This layer downsamples the output of the convolutional layer by taking the maximum value within a window of size 3. It helps to reduce dimensionality and makes the model more robust to small variations in word order.

Conv1D (128 filters, kernel size 3, ReLU): Another convolutional layer, similar to the first one. It further refines the features learned in the previous convolutional layer.

MaxPooling1D (pool size 3): Another max pooling layer for downsampling and robustness.

Conv1D (128 filters, kernel size 3, ReLU): A third convolutional layer, continuing the feature extraction process.

GlobalMaxPooling1D: This layer takes the maximum value across the entire feature map for each of the 128 filters. It effectively summarizes the most important features detected by the convolutional layers. This produces a fixed-length vector regardless of the input text length.

Dense (128 neurons, ReLU): A fully connected (dense) layer with 128 neurons and ReLU activation. It further processes the features extracted by the convolutional layers.

2.1.3 SVM

Support Vector Machines (SVMs) are supervised learning algorithms used for classification and regression problems.The main goal of SVMs is to find the ”optimal” hyperplane that best separates the data of different classes.”Optimal” means that the hyperplane maximizes the margin between the closest data points of each class.In other words, it tries to create the widest possible ”road” between the classes.SVMs are particularly useful when the data has many dimensions (many ”features”). They can also handle data that is not linearly separable by using a ”kernel trick”.This means that they can transform the data into a higher dimensional space where it is easier to find a separating hyperplane. In our case the kernel chosen is the linear(instead of poly or sigmoid) to improve the training times.

2.2 Encoding

ML and DL models inherently operate on numerical data. Consequently, textual input must be converted into a numerical format to be processed. Encoding fulfills this fundamental need, translating linguistic units (words, phrases, documents) into numerical vectors.

However, this transformation is not merely a format conversion. Encoding aims to capture the semantic and syntactic properties of the text in a numerical form interpretable by the models. Ideally, similar numerical vectors in the vector space should correspond to linguistic units with related meanings, enabling the models to learn complex relationships between words and phrases.

2.2.1 TF_IDF

TF-IDF is an ingenious technique that allows us to assign a numerical value to the importance of a word within a text, taking into account how common or rare that word is in a larger set of texts. Imagine you have a text and you want to understand which are the most significant words, those that best represent it. TF-IDF helps you precisely in this. It considers two factors: how often a word appears in the text, because if a word repeats itself a lot, it is likely that it is important, and how rare that word is in other texts, because if a word is very common in many texts, it is probably not very useful to distinguish a specific text. By combining these two factors, TF-IDF assigns a score to each word. The higher the score, the more the word is considered important for that specific text. This technique is very useful in various fields, such as information retrieval, where it helps to understand which web pages are most relevant to your search, automatic text summarization, where it can identify the most important keywords of a text to create a short but meaningful summary, and text classification, where it can help to understand what a text is about and which category it belongs to. In essence, TF-IDF is a powerful tool that allows us to understand which are the most important words in a text compared to a set of other texts, opening the way to many useful applications in the field of natural language processing.

2.2.2 Encoding Layer

In order to build our encoding layer, we first need to construct our embedding matrix. The embedding matrix is a matrix with as many rows as there are words in the vocabulary (or fewer for more efficient computation) and as many columns as the dimension of the encoding used (in this case glove50d, therefore 50).Once we have constructed our embedding matrix, we use the appropriate TensorFlow function, passing the matrix to it along with other values.

Regarding the SVM model, I created a pipeline to directly input the data. To create the encoder, I used the scikitlearn function "TfidfVectorizer" for tf_idf, setting some parameters to improve training times such as the word limit to 20,000 and the use of only unigrams.

$$\begin{array}{l} \text{hello} \\ \text{there} \\ \text{texas} \\ \text{world} \\ \ldots \end{array} \begin{bmatrix} 12 & 45 & 43 & 26 & 78 & 532 & \ldots \\ 43 & 25 & 778 & 43 & 53 & 78 & \ldots \\ 34 & 56 & 23 & 12 & 56 & 74 & \ldots \\ 342 & 54 & 23 & 5 & 7 & 423 & \ldots \\ \ldots & & & & & & \end{bmatrix}$$

Figura 2.1: Example of embedding matrix.

CAPITOLO 3

Experiment

3.1 Esploratory Data Analisys

3.1.1 Heatmap

First, we analyze our dataset to get useful information from it.Our objective is to investigate the correlation among the different features within our dataset. The correlation matrix is presented below: It can be observed from the image that the features 'Insult' and 'Obscene' are correlated.

3.1.2 Most frequent words

Now I would like to understand which are the most frequent words in the dataset. To do this, I split the words in the dataset and counted them using the Counter function from the type library. I extracted the 100 most frequent ones and created the following word cloud: As one might expect, the most probable words are adjectives and articles.This is a common observation when analyzing text data without removing stop

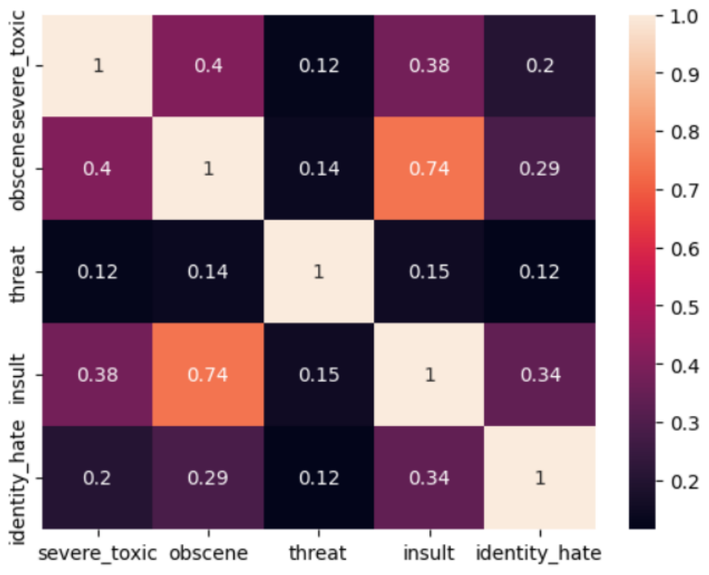

Figura 3.1: Heatmap Train Data.

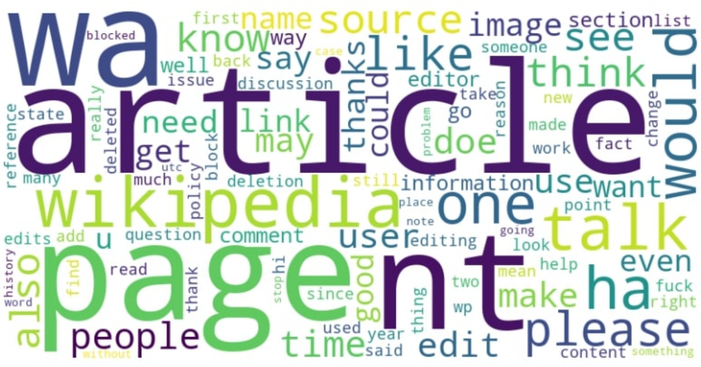

Figura 3.2: Frequencies Words.

words. Stop words are common words that are often removed during natural language processing (**NLP**) tasks because they don't carry much meaning on their own. In English, examples of stop words include "the," "a," "is," "are," and "and.To remove the stop words, we will use the function of the same name from the **NLTK** library. The result is in the figure 1.3. However, this graph also doesn't give us much information. For this reason, we will remove all non-toxic comments from the dataset to see if anything changes.(Figure 1.4) We can see the changes in the fact that a good portion of the most frequent words are offensive words.

3.2 Data Cleaning

In the field of natural language processing (**NLP**), text datasets are a fundamental resource for training machine learning models and analyzing textual content. However, raw text data is often "dirty" and requires careful cleaning to ensure the reliability and effectiveness of the analyses. Data cleaning is a crucial process that aims to identify and correct or remove errors, inconsistencies, and noise present in the data. In

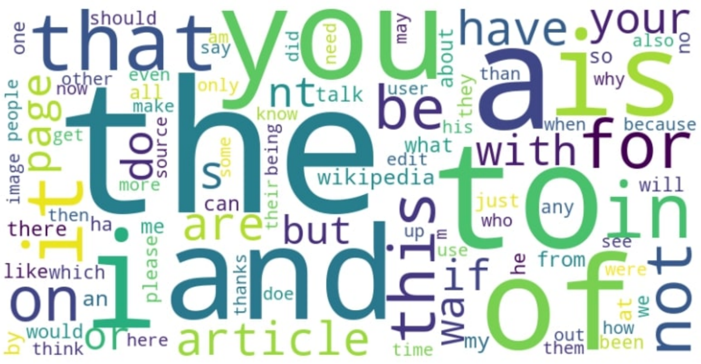

Figura 3.3: Frequencies Words without StopWords.

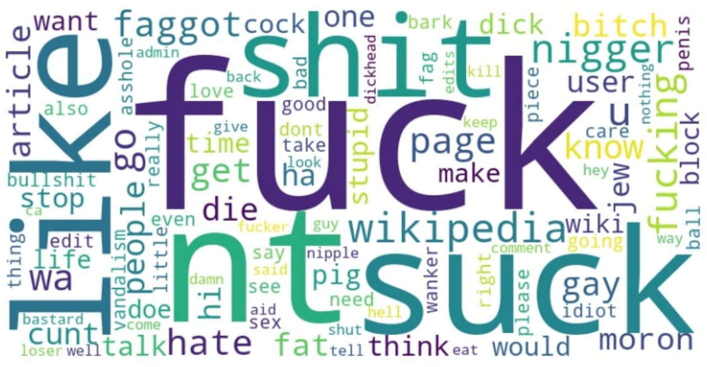

Figura 3.4: Frequencies Words only Toxic Comment.

this introduction, we will explore the main data cleaning operations applicable to text datasets.

1. **Removal of Special Characters and HTML Codes:**Texts may contain special characters (such as symbols, non-standard punctuation, etc.) and HTML codes (if the data comes from web pages) that are not relevant for text analysis. Removing these elements is fundamental step to ”clean” the text.

2. **Text Normalization:**Text normalization consists of transforming the text into a standard form to facilitate analysis. Common normalization operations include:**Lowercasing**,**Stop word removal**,**Stemming and Lemmatization**

3. **Tokenization:**Tokenization is the process of dividing the text into smaller units, called tokens, which can be words, phrases, or other elements. Tokenization is a fundamental step for many NLP tasks.

In particular, in graphs 1.5, we can see the distribution of the size of the words in the comments before the data cleaning.

We can also see from this graph that the majority of the dataset remains under 100 words per comment, as the 75th percentile is 69 words. To achieve good model performance, we need to decide how to handle these outliers.

3.3 Trainings

Once the exploratory data analysis and dataset cleaning have been done. I saved the ”cleaned” dataset in csv called data_cleaned.csv so that I can import it into the notebook where we train the models. In addition to the dataset, we also import the tokenizer used in cleaning so that it is ready for use. The notebook then proceeds in the following phases:

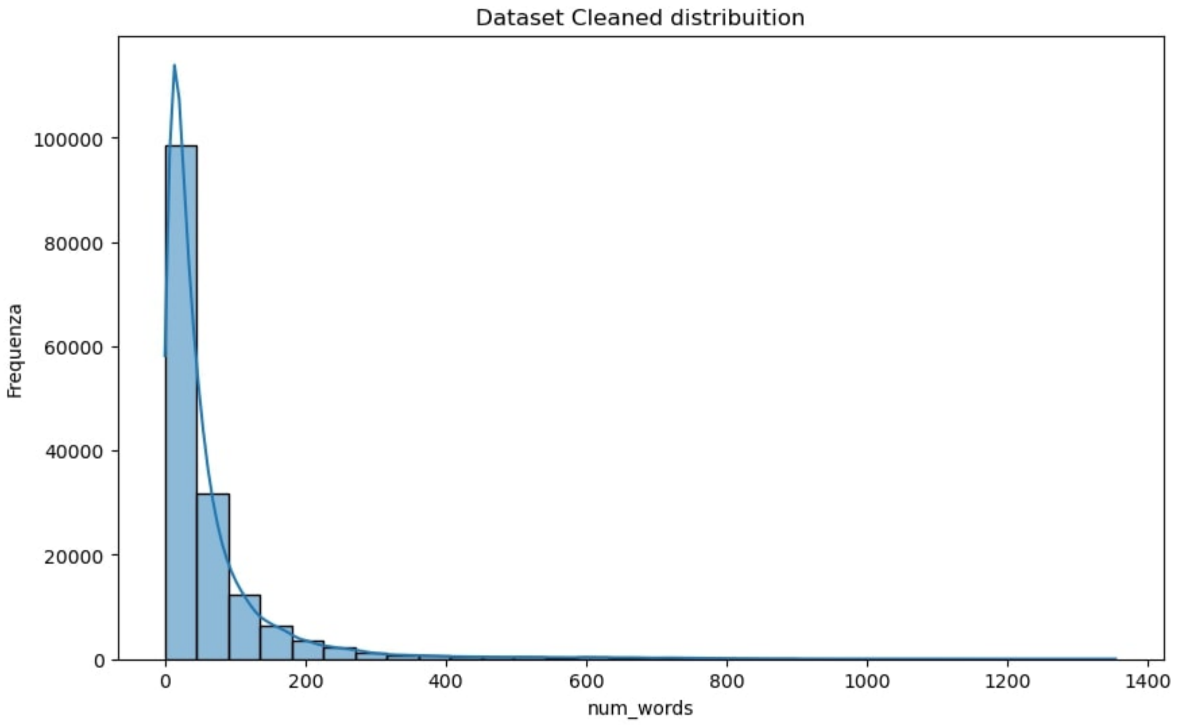

Figura 3.5: Distribution of Word occurencies.

- **Loading the GloVe Model:** I previously downloaded the pre-trained model from Stanford's site and uploaded the 50-vector version. Saving the words in a dictionary with key and the vectors of the encoded words as values.
- **Embedding Layer and Training Constant:** As already described we create the embedding layer with the matrix built from the GloVE vectors. We also set some constants useful in training such as the sequence size (in our case 100 words), the maximum vocabulary size (Vocab_size) the embedding size (that of the Glove model) and the training parameters batch_size, epochs.
- **CNN, MLP, SVM:** the parts where we declare the various models. I put them in methods so that I don't have to re-declare them every time.
- **K-fold Validation:** Here the actual training takes place apart from the test examples already done before. In particular we use the StratifiedKFold to keep the classes in their proportions (even if the dataset is unbalanced) to avoid overfitting. During training all the models (trained with the various folds) are saved in the models folder. The models are also measured and the metrics saved in a variable. (so that we can create the data graphs later)
- **Valutaion:** In this section the various models are compared.

The results of the various models are shown in the figures 3.6,3.7,3.8:

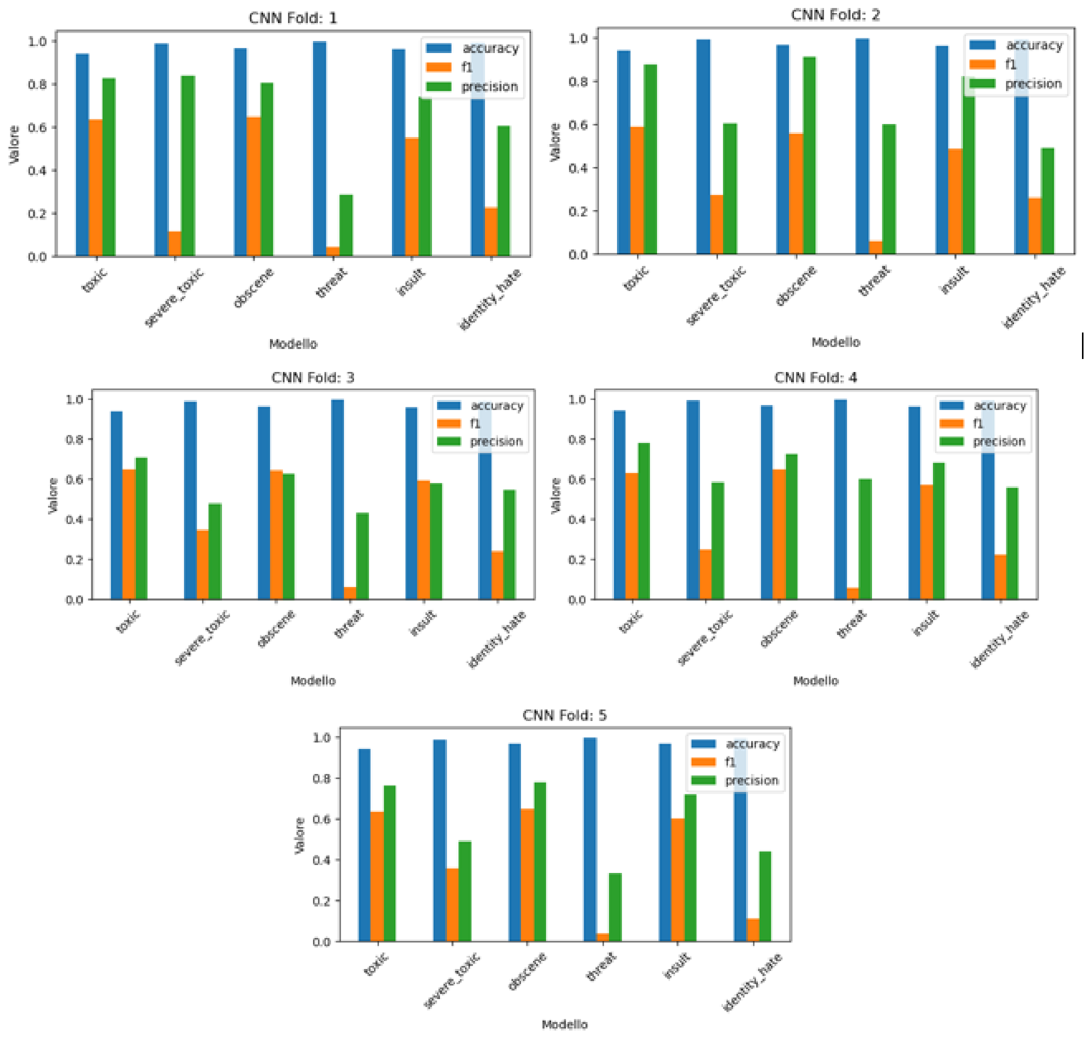

Figura 3.6: Enter Caption.

3.3.1 Results

As we can see from graphs 3.6 to 3.8, convolutional neural networks generally obtain better results than multi-layerperceptron and SVM. Furthermore, in terms of inference times they are second only to MLPs. The TF_IDF approach combined with SVM is slightly less performing than the others and above all the most expensive in terms of training. Furthermore, I also tried to predict the dataset "test.csv" that is in the downloaded folder. But there are no results from it (I also tried to search on the internet) however I attached the file in the project anyway.

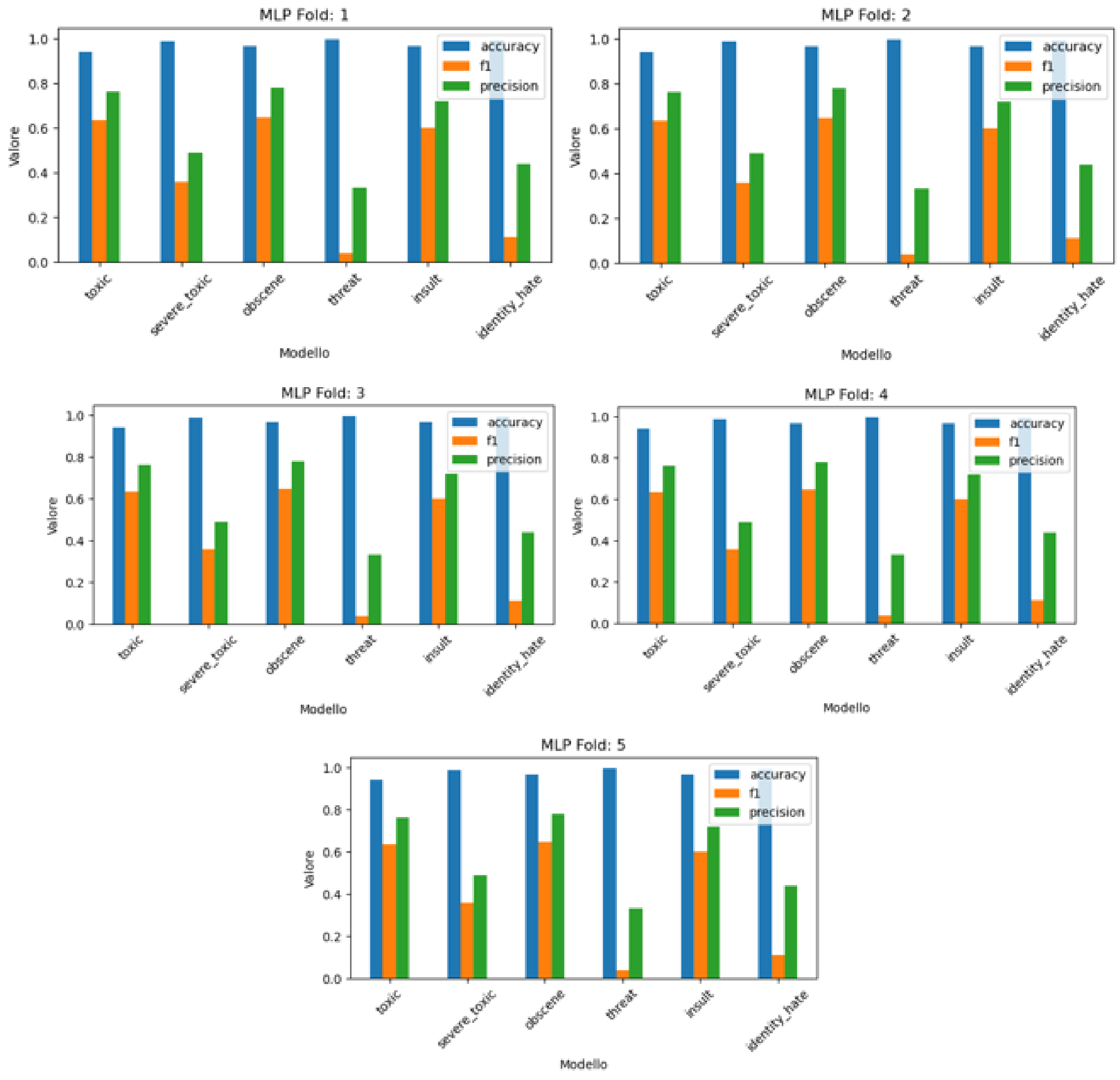

Figura 3.7: Enter Caption.

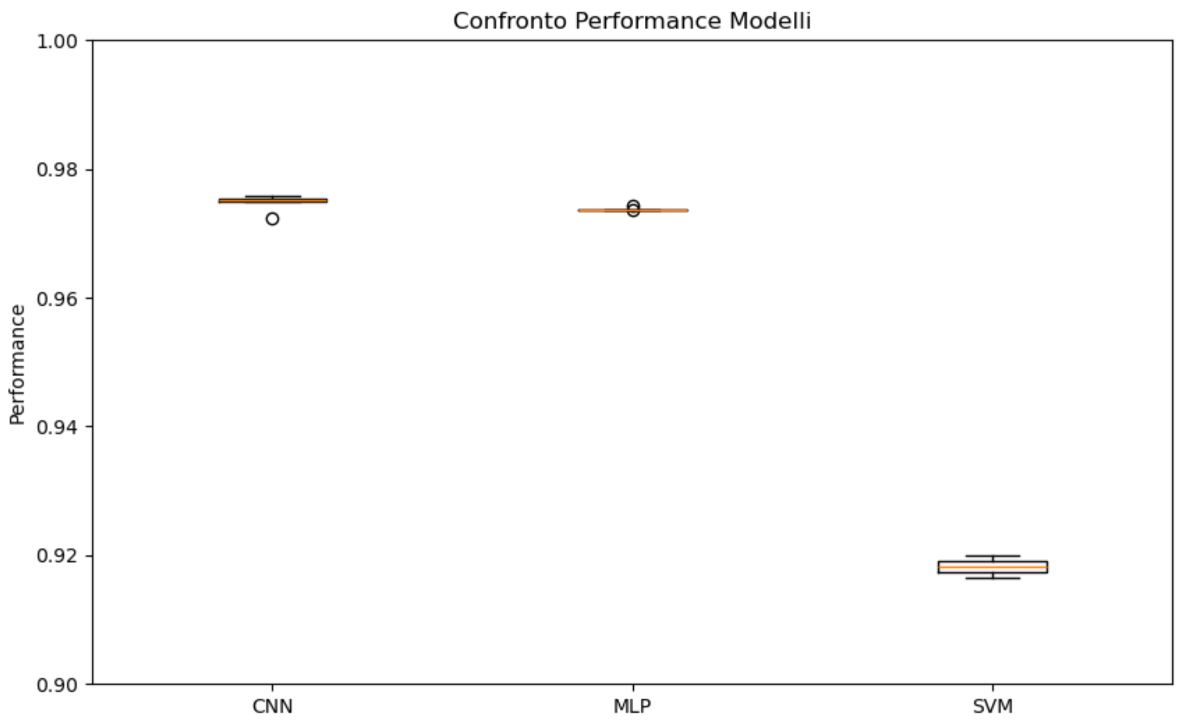

Figura 3.8: Enter Caption.