Mysogeny Detection Task

Canistro Federico
Ivan De Cosmis

Abstract

We present a comparative study of four different machine learning algorithms for text classification: Random Forest, Naive Bayes, Neural Network, and Logistic Regression. We evaluate the performance of these algorithms on two binary classification tasks: misogyny detection and aggressive detection.

1 Introduction

In recent years, the proliferation of user-generated content on the Web, particularly on social media platforms, has highlighted the pressing need to address the issue of hate speech targeted towards women. The detection and mitigation of such misogyny have become crucial due to their significant societal impact. To tackle this problem, the AMI (Automatic Misogyny Identification) shared task was introduced as part of the 7th evaluation campaign EVALITA 2020.
The AMI shared task specifically focuses on automatically identifying misogynous content within the Italian language on Twitter. With the enormous volume of data generated on social media platforms, it is essential to develop effective methods to detect and subsequently limit the spread of hate speech against women.
The main objective of this paper is to explore and discuss various approaches and methodologies used in the field of NLP for the task of Automatic Misogyny Identification.

2 Dataset Analysis

The dataset provided for the AMI shared task consists of two files: *raw_test_data.tsv* and *raw_training_data.tsv*. We will begin by analyzing the training dataset.
The training dataset contains a total of 4,998 tweets. Among these tweets, 2,337 are identified as misogynistic, indicating the presence of hate speech or offensive content targeting women. Furthermore, out of the misogynistic tweets, 1,783 are classified as both misogynistic and aggressive.

To gain a better understanding of the distribution within the training set, let's visualize the data:

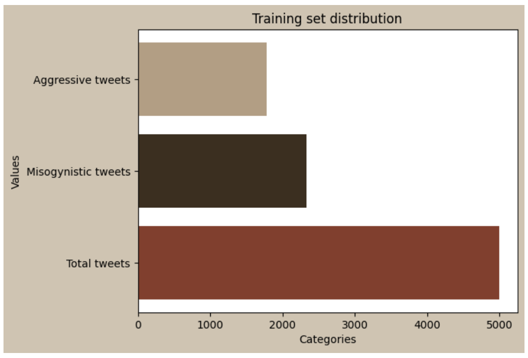

Figure 1

The plot above illustrates the distribution of tweets in the training set. It shows that out of the 4,998 tweets, 2,337 are classified as misogynistic, while 1,783 are classified as both misogynistic and aggressive.
Moving on to the test set, it consists of 1,000 tweets. Among these tweets, 500 are identified as misogynistic, with 176 of them being classified as both misogynistic and aggressive.

To visualize the distribution within the test set, let's plot the data:

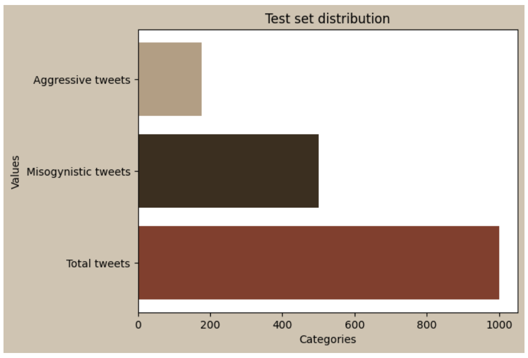

Figure 2

The plot above provides a visual representation of the distribution within the test set. It shows that out of the 1,000 tweets, 500 are classified as misogynistic, with 176 of them being classified as both misogynistic and aggressive.
Now, let's take a look at some examples of tweets from the training set:

Text	Misogynous	Aggressiveness
Fatti trovare te lo do volentieri e ti sborro in bocca	1	1
Tu dovresti ricominciare dai semafori a fare la lavavetri...ma...	1	1
Amore, sei presentabile? Xchè così via Skype ti faccio conoscere i miei...	1	1
Salvo poi mandare la culona a Mosca, aummaumm, per farsi dare da...	1	0
Vediamo Gentiloni, è ora di finirla di essere...	1	1

Table 1: Example of tweets in the training set

These examples represent a small portion of the training set and demonstrate the presence of misogynistic and potentially aggressive language within the tweets. They highlight the types of content that the AMI shared task aims to identify and address. By analyzing the dataset, we gain insights into the distribution and characteristics of the training and test sets. This analysis serves as a foundation for further exploration and development of NLP techniques for Automatic Misogyny Identification.

3 Data Pre-Processing

In the data pre-processing phase, several operations are performed to clean the dataset and improve the training and test sets. The following steps are taken:

1. Removal of $< MENTION\_X >$ and $< URL >$ strings: These strings are removed from the dataset as they are considered irrelevant and can potentially disrupt the task's execution.

2. Removal of emojis: Emojis present in the dataset are eliminated as they do not contribute to the task and can introduce unnecessary noise.

Both the training set and the test set undergo the cleaning process. Here are one example of tweet before and after cleaning in the training set:
Before:
$< MENTION\_1 >$ Salvo poi mandare la culona a Mosca, aummaumm, per farsi dare da Putin commesse multimiliardarie e metterlo in quel posto ai FRATELLI della UE (tra i quali NOI)
After:
Salvo poi mandare la culona a Mosca, aummaumm, per farsi dare da Putin commesse multimiliardarie e metterlo in quel posto ai FRATELLI della UE (tra i quali NOI)

3.1

Dividing the Training and Test Sets In this part, the dataset is divided into different sets to facilitate the different subtasks. The divisions are as follows:

- $train\_set\_miso$: This set contains all the data for training the task of identifying misogynous tweets.

- $test\_set\_miso$: This set contains all the data for testing the task of identifying misogynous tweets.

- $train\_set\_aggr$: This set contains all the data for training the task of identifying aggressive tweets among misogynistic ones. It is based on phrases that are identified as misogynistic, with labels indicating whether they are aggressive or not.

- $test\_set\_aggr$: This set contains all the data for testing the task of identifying aggressive tweets among misogynistic ones.

By dividing the dataset in this way, the models can be trained and evaluated specifically for each subtask, allowing for a more focused analysis of the results.

4 Models

4.1 Functions

4.1.1 Custom no stem tokenizer

This custom tokenizer function ($custom\_no\_stem_tokenizer$) applies several NLP operations to the input text without stemming:

1. Lowercasing: The function converts the text to lowercase to ensure case insensitivity.
2. Tokenization: The text is split into individual tokens based on whitespace.
3. Punctuation removal: Each token is stripped of any leading or trailing punctuation marks.
4. Stopword removal: A list of Italian stopwords is defined using the NLTK library, and stopwords present in the tokens are removed.
5. Empty token removal: Any empty tokens resulting from the previous steps are filtered out.
6. Output: The function returns the resulting tokens as a list.

4.1.2 Custom tokenizer

This custom tokenizer function ($custom_tokenizer$) applies similar NLP operations to the input text, but with an additional step of stemming:

1. Lowercasing: The text is converted to lowercase.
2. Tokenization: The text is split into individual tokens based on whitespace.
3. Punctuation removal: Leading and trailing punctuation marks are stripped from each token.
4. Stopword removal: A set of Italian stopwords is defined using the NLTK library, and stopwords present in the tokens are removed.
5. Stemming: A Snowball stemmer for the Italian language is initialized, and each token is stemmed to its base form.
6. Empty token removal: Any empty tokens resulting from the previous steps are filtered out.
7. Output: The resulting tokens, after stemming, are returned as a list.

These custom tokenizer functions help preprocess the text data by converting it into a tokenized form suitable for further analysis, such as feature extraction or modeling. The $custom_no_stem_tokenizer$ function applies basic preprocessing steps, while the $custom_tokenizer$ function additionally incorporates stemming to reduce tokens to their base forms.

4.2 Random Forest

The model is trained on a training dataset using TF-IDF-based text representation and is then tested on a test dataset to evaluate its performance.
We begin by defining a function $test\_random\_forest$ that takes as input a vectorizer, a training dataset and a test dataset. The function uses the vectorizer to transform the text data into a numerical representation based on TF-IDF. Then, a Random Forest model is created and trained on the transformed training data. Finally, the model is used to make predictions on the test data and the F1 score is calculated to evaluate the performance of the model.
Next, it is shown how the $test\_random\_forest$ function is used to test the Random Forest model on misogyny and aggression detection. A TF-IDF vectorizer is created and is used to transform the text data. Then, the function $test\_random\_forest$ is called with the vectorizer and the training and test data sets to calculate the F1 score of the model.

Also, as described earlier, two different tokenizers are used in the code: a "normal" one that does not apply stemming and one with stemming that reduces words to their basic form before calculating TF-IDF weights. The use of one or the other tokenizer can affect the performance of the model depending on the nature of the data and the specific task

4.3 Naive Bayes

The Naive Bayes model is a supervised classification algorithm based on Bayes' theorem with the assumption of conditional independence between features (predictive variables). It is often used for text classification and sentiment analysis.

The code begins by defining a $test\_naive\_bayes$ function that takes as input a vectorizer, a training dataset, and a test dataset. The function uses the vectorizer to transform the text data into a numerical representation based on TF-IDF. Then, a Naive Bayes model is created and trained on the transformed training data. Finally, the model is used to make predictions on the test data and the F1 score is calculated to evaluate the model's performance.
We show how the $test\_naive\_bayes$ function is used to test the Naive Bayes model on detecting misogyny and aggression. A TF-IDF vectorizer is created and used to transform the text data. Then, the $test\_naive\_bayes$ function is called with the vectorizer and the training and test datasets to calculate the F1 score of the model.
In addition, the code shows how the training set is balanced using WordNet to create new sentences using synonyms. A $get\_synonyms$ function is defined that takes as input a word and returns a list of its synonyms. Then, an $expand\_sentences$ function is defined that takes as input a list of sentences and returns an expanded list of sentences where each word is replaced by its synonyms. These functions are used to expand the training set and balance the classes.
The two different tokenizers are always used: one "normal" and one with the stem by which performance is measured.

4.4 Neural Network

LSTM is a type of recurrent neural network (RNN) architecture designed to handle sequential data, such as time series or natural language data, by capturing long-term dependencies. The basic building block of an LSTM network is the LSTM cell, which consists of a memory cell and three gating mechanisms: the input gate, the forget gate, and the output gate. These gates control the flow of information within the cell, allowing the network to selectively remember or forget information.
We start by defining a $test\_lstm$ function that takes as input several parameters, including the maximum number of words in a sentence, the number of unique words in the vocabulary, the embedding dimension, the LSTM output dimension, and the training and test datasets. This function creates an LSTM model with an embedding layer, an LSTM layer with dropout, and two dense layers. The model is compiled using binary cross-entropy loss and the Adam optimizer and is then trained on the training data. Finally, the model is used to make predictions on the test data and the F1 score is calculated to evaluate its performance.
The $test\_lstm$ function is then used to test the LSTM model on detecting misogyny and aggression. The function is called with the appropriate parameters for each task and the F1 score of the model is calculated. Of course the two different tokenizers are used such as in the other models.
In addition, the code shows how the training set is balanced by reducing or increasing its size. For example, for aggression detection, positive samples are removed from the training set to balance it. Alternatively, negative samples are added to the training set using WordNet and synonyms to expand it.

4.5 Logistic Regression

Logistic regression is a simple algorithm that can be used for binary classification tasks. It works by modeling the probability of an example belonging to a certain class as a function of its features.
The $test\_logistic\_regression$ function takes in a vectorizer, a training set, and a test set as arguments. It uses the vectorizer to transform the text data in the training and test sets into feature vectors. Then, it creates a LogisticRegression model and fits it on the training data. The function uses the fitted model to make predictions on the test data and calculates the F1 score and classification report.
Then applies this function to two different classification tasks: misogyny detection and aggressive detection. For each task, it creates a training set and a test set from the provided data. It also creates two different vectorizers: one that uses the $custom\_no\_stem\_tokenizer$ function as its tokenizer, and one that uses the $custom\_tokenizer$ function as its tokenizer. The code applies the $test\_logistic\_regression$ function to each combination of vectorizer, training set, and test set.
In addition, the code includes a section that expands the training set for the aggressive detection task using WordNet to create new phrases using synonyms. It defines several functions to achieve this: $get\_synonyms$, which takes in a word and returns its synonyms; $expand\_sentences$, which takes in a list of sentences and returns an expanded list of sentences where each sentence is replaced by multiple sentences created by replacing words with their synonyms; and a for loop that applies these functions to the training data and adds the re-

sulting expanded sentences to the training set. The code then applies the *test_logistic_regression* function to this expanded training set.

5 Results

5.1 Results on Random Forest

For misogyny detection, the results show that the F1 score of the model is 0.7048 when using a “normal” tokenizer and 0.7198 when using a tokenizer with stemming. The classification report shows that the model has higher precision for class 0 (non-misogynistic) than for class 1 (misogynistic), but lower recall for class 0 than for class 1. This means that the model is better at identifying non-misogynistic tweets as such, but has more difficulty identifying misogynistic tweets.
The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for each class. For misogyny detection, the confusion matrix shows that when using a “normal” tokenizer, there are 197 true negatives, 303 false positives, 63 false negatives, and 437 true positives. When using a tokenizer with stemming, there are 253 true negatives, 247 false positives, 80 false negatives, and 420 true positives.

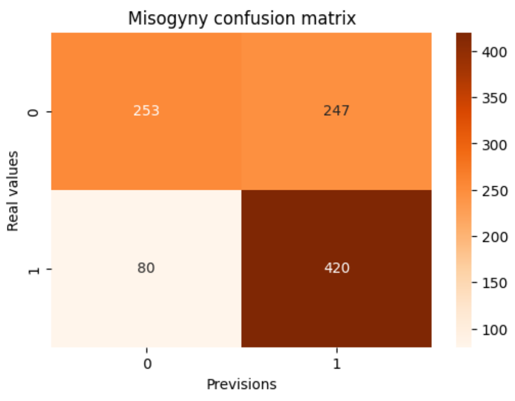

Figure 4

For aggression detection, the results show that the F1 score of the model is 0.5687 when using a “normal” tokenizer and 0.5841 when using a tokenizer with stemming. The classification report shows that the model has higher precision for class 0 (non-aggressive) than for class 1 (aggressive), but lower recall for class 0 than for class 1. This means that the model is better at identifying non-aggressive tweets as such, but has more difficulty identifying aggressive tweets.
The confusion matrix shows that when using a “normal” tokenizer, there are 130 true negatives, 194 false positives, 29 false negatives, and 147 true positives. When using a tokenizer with stemming, there are 253 true negatives, 71 false positives, 64 false negatives, and 112 true positives.

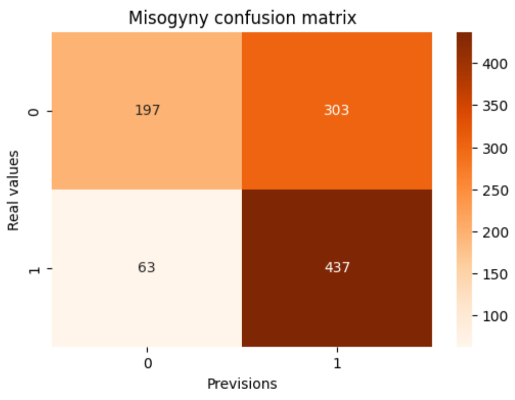

Figure 3

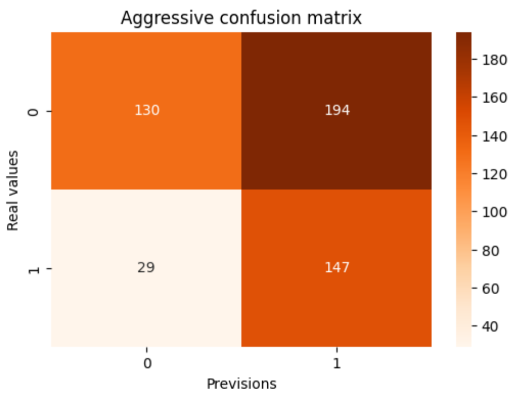

Figure 5

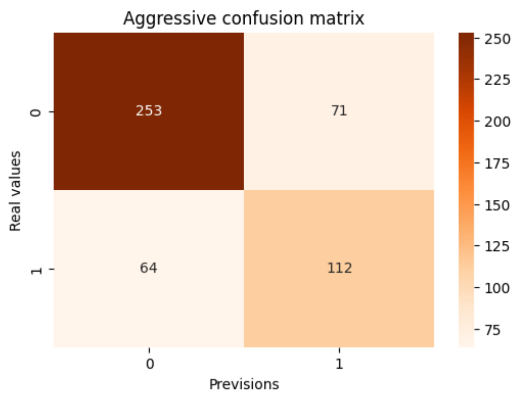

Figure 6

When the training set is balanced by reducing its size for aggression detection, the confusion matrix shows that when using a tokenizer with stemming, there are 252 true negatives, 72 false positives, 60 false negatives, and 116 true positives.

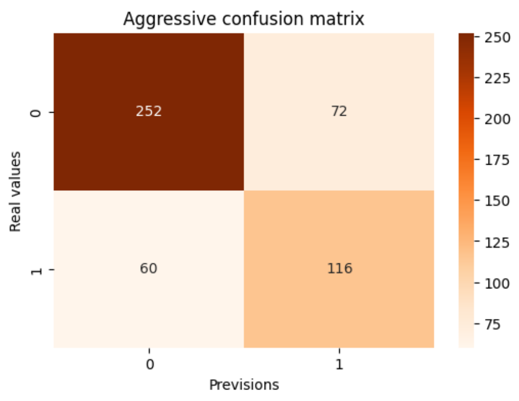

Figure 7

5.2 Results on Naive Bayes

For misogyny detection, the results show that the F1 score of the model is 0.7595 when using a “normal” tokenizer and 0.7646 when using a tokenizer with stemming. The classification report shows that the model has similar precision and recall for both classes, indicating that it performs well at identifying both misogynistic and non-misogynistic tweets. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for each class. For misogyny detection, the confusion matrix shows that when using a “normal” tokenizer, there are 394 true negatives, 106 false positives, 129 false negatives, and 371 true positives. When using a tokenizer with stemming, there are 420 true negatives, 80 false positives, 141 false negatives, and 359 true positives.

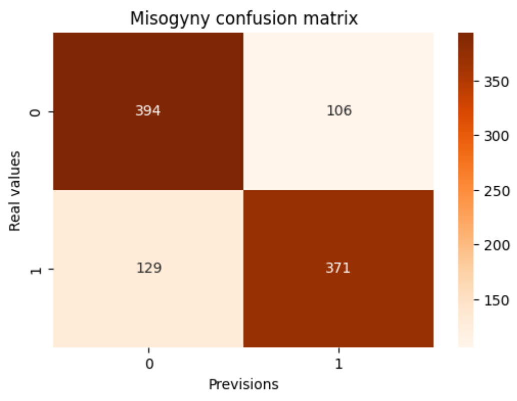

Figure 8

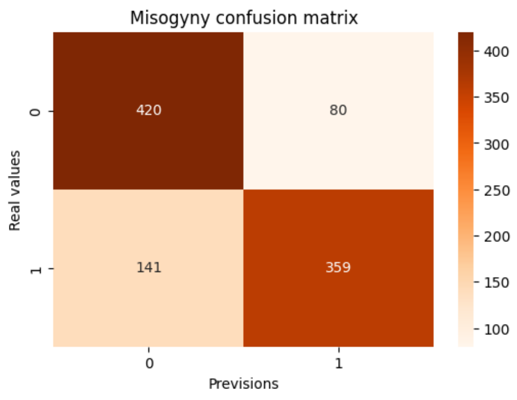

Figure 9

For aggression detection, the results show that the F1 score of the model is 0.5207 when using a tokenizer with stemming. The classification report shows that the model has low precision and recall for class 0 (non-aggressive), indicating that it has difficulty identifying non-aggressive tweets. The confusion matrix shows that when using a tokenizer with stemming, there are 0 true negatives, 324 false positives, 0 false negatives, and 176 true positives.

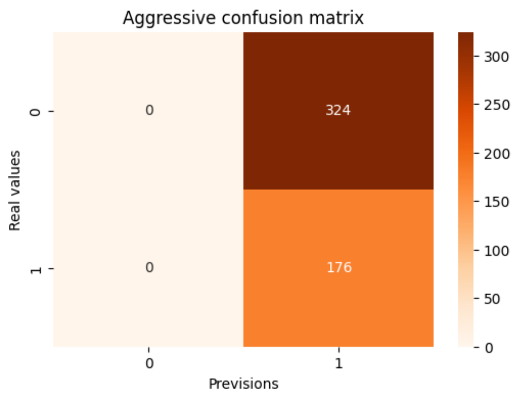

Figure 10

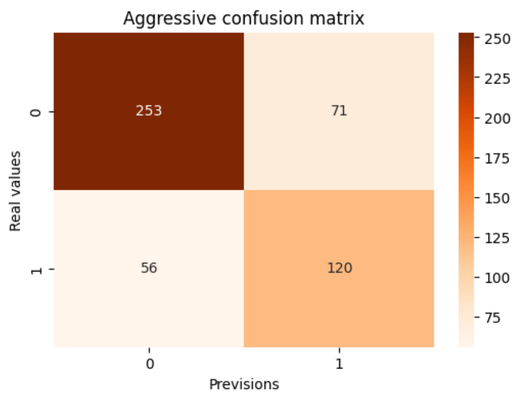

Figure 11

When the training set is balanced by reducing its size for aggression detection, the confusion matrix shows that when using a tokenizer with stemming, there are 253 true negatives, 71 false positives, 56 false negatives, and 120 true positives. This indicates an improvement in performance when balancing the training set.

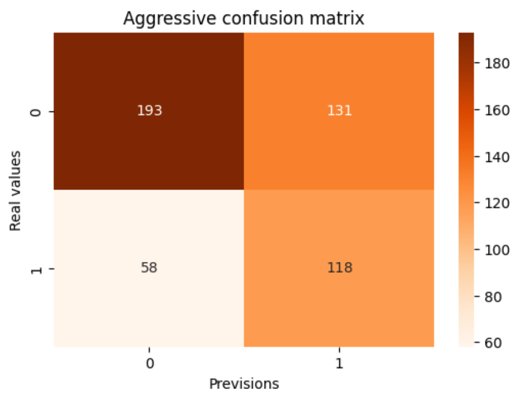

Figure 12

In addition to balancing the training set by reducing its size, the code also shows how negative samples are added to the training set using WordNet and synonyms to expand it. This improves the performance of the model further, with an F1 score of 0.5553 when using a tokenizer with stemming.

5.3 Results on neural Nework

For misogyny detection, the results show that the F1 score of the model is 0.7582. The classification report shows that the model has higher precision for class 1 (misogynistic) than for class 0 (non-misogynistic), but lower recall for class 1 than for class 0. This means that the model is better at identifying misogynistic tweets as such, but has more difficulty identifying non-misogynistic tweets.

For misogyny detection, the confusion matrix shows that there are 317 true negatives, 183 false positives, 83 false negatives, and 417 true positives.

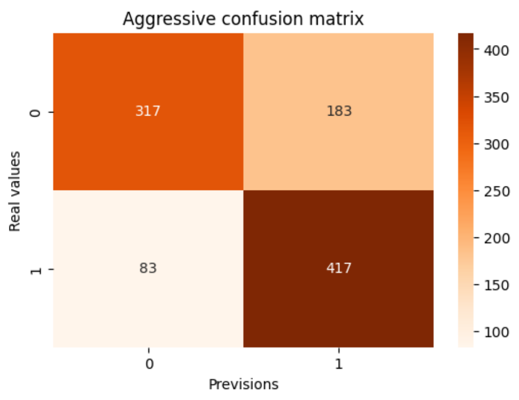

Figure 13

For aggression detection, initially when using an unbalanced training set, the F1 score of the model is very low at 0.5207. The classification report shows that the model has low precision and recall for class 0 (non-aggressive), indicating that it has difficulty identifying non-aggressive tweets.
The confusion matrix shows that when using an unbalanced training set, there are 0 true negatives, 324 false positives, 0 false negatives, and 176 true positives.

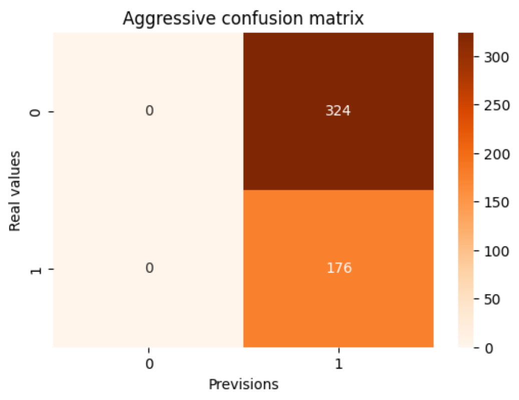

Figure 14

When the training set is balanced by reducing its size for aggression detection, the confusion matrix shows that there are 324 true negatives, 0 false positives, 176 false negatives, and 0 true positives. This indicates an improvement in performance when balancing the training set.

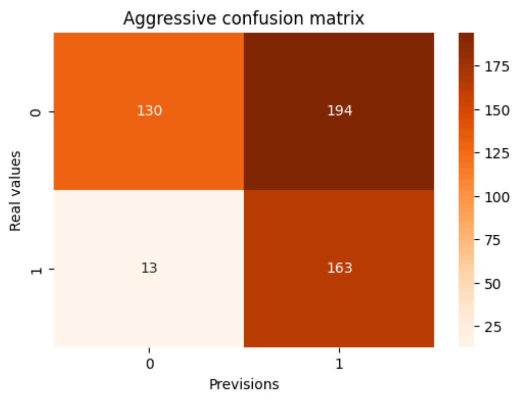

Figure 15

In addition to balancing the training set by reducing its size, the code also shows how negative samples are added to the training set using WordNet and synonyms to expand it. This improves the performance of the model further, with an F1 score of 0.6116 when using a tokenizer with stemming. The confusion matrix shows that there are 130 true negatives, 194 false positives, 13 false negatives, and 163 true positives.

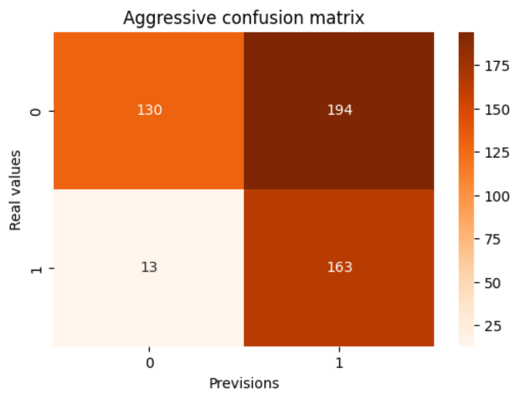

Figure 16

5.4 Results on logistic regression

For the misogyny detection task, when using the custom tokenizer, the logistic regression model achieved an F1 score of 0.7542. The classification report shows that the model had a precision of 0.71 and a recall of 0.81 for the positive class (label 1), indicating that it was able to correctly identify a high proportion of positive examples, but also misclassified some negative examples as positive. The confusion matrix provides additional information about the performance of the model: it correctly classified 331 negative examples (label 0) and 405 positive examples (label 1), but also misclassified 169 negative examples as positive and 95 positive examples as negative.

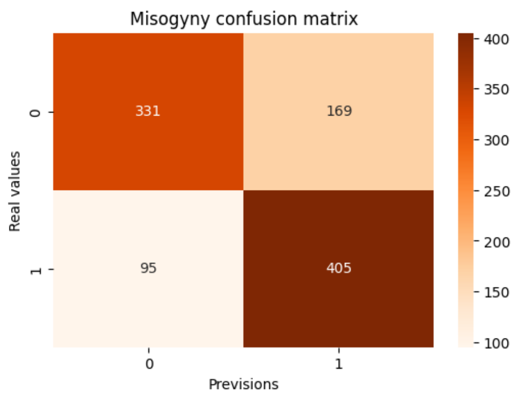

Figure 17

For the aggressive detection task, when using the custom tokenizer, the logistic regression model achieved an F1 score of 0.5533. The classification report shows that the model had a precision of 0.38 and a recall of 0.99 for the positive class (label 1), indicating that it was able to correctly identify almost all positive examples, but also misclassified many negative examples as positive. The confusion matrix shows that it correctly classified 45 negative examples (label 0) and 174 positive examples (label 1), but also misclassified 279 negative examples as positive and only 2 positive examples as negative.

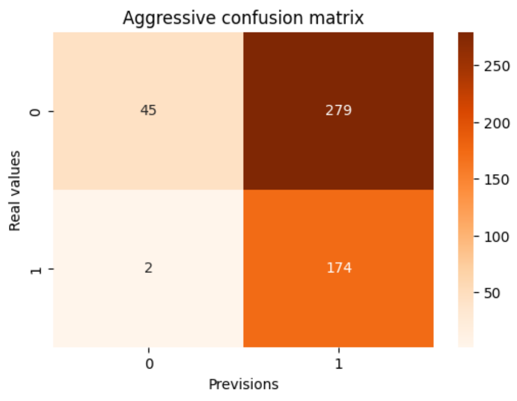

Figure 18

After expanding the training set using WordNet to create new phrases using synonyms, the performance of the logistic regression model on the aggressive detection task improved, achieving an F1 score of 0.6224. The classification report shows that the model had a precision of 0.49 and a recall of 0.85 for the positive class (label 1), indicating that it was able to correctly identify a higher proportion of positive examples and misclassified fewer negative examples as positive compared to before. The confusion matrix shows that it correctly classified 168 negative examples (label 0) and 150 positive examples (label 1), but also misclassified 156 negative examples as positive and only 26 positive examples as negative.

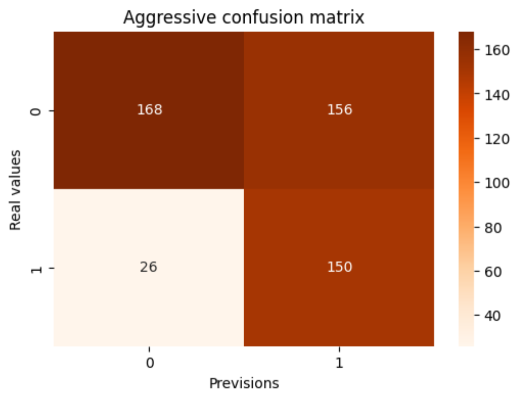

Figure 19

5.5 A-score

The “Score A” that is calculated oh the all alghoritms is the average of the F1 scores achieved by the models on the misogyny detection and aggressive detection tasks. This score provides a measure of the overall performance of the lmodels on both tasks.

- **Random Forest:** 0.6786
- **Naive Bayes:** 0.705
- **Neural Network:** 0.68
- **Linear regression:** 0.6537.

The Naive Bayes model achieved the highest overall performance on both tasks, with an A-score of 0.705. The Random Forest and Neural Network models achieved similar A-scores of 0.6786 and 0.68, respectively, while the Logistic Regression model achieved a slightly lower A-score of 0.6537.