UNIVERSITY OF PISA

DEPARTMENT OF INFORMATICS

TEXT ANALYTICS

PERSUASION TECHNIQUES CLASSIFICATION: RESULT OF EXPLORATIONS, ANALYSIS AND CONSIDERATIONS

GROUP 4

Diego Borsetto(627692)
Giulio Canapa(568649)
Sara Quattrone (646438)
Simona Sette (544298)

ACADEMIC YEAR 2022/2023

Contents

1. Project Introduction

The project goal takes inspiration from the subtask3 of the shared task on SemEval2023 called "Detecting the genre, the framing, and the persuasion techniques in online news in a multilingual setup". The original SemEval subtask is a multi-labeling classification problem whose goal is to assign one or more communication persuasion techniques to paragraphs extracted from news articles. For this project it was decided to set up a multi-class classification task: given the paragraphs, the goal is to attribute individual persuasion techniques to individual paragraphs.

Testo paragrafo	Tecnica di persuasione risultante
Geneva - The World Health Organisation chief on Wednesday said a deadly plague epidemic appeared to have been brought under control in Madagascar, but warned the next outbreak would likely be stronger.	Doubt

Fig 1. Example of the classification goal.

The reasons behind the choice of this experimental task is the interest in investigating the possibility to automatically identify communicative persuasion techniques through the textual linguistic components. Persuasive communication is also responsible for narrative manipulations, impacting and unconsciously influencing public opinion through nowadays media: the automatic identification of these techniques can be a game-changer in terms of misinformative news identification and improvement of the information quality.

2. Dataset Generation

The provided original data were made available through text files that were manipulated in order to generate a dataset. The total number of unique original labels was 23 and the total number of labeled paragraphs (with one or more labels) was 3760. Considering our aim to achieve a multi-class-single-label classification, it was chosen to not consider the paragraphs with more labels, resulting in a first skimmed selection of 2308 paragraphs and 19 labels; the 3 excluded labels always appeared in combination with the others and for this reason went lost.
At this point a merge between the datasets containing the labels and textual paragraph was made through the article and paragraph ids combination, returning a final dataset containing 19 distinct persuasion techniques and 2217 paragraph rows from 416 different newspaper articles, where a single label corresponds to a single paragraph.
The reason for the loss of 91 records, compared to the previous skimming, consists in the presence of paragraphs with an assigned label but not a corresponding text, probably lost due to human error.
A first brief exploration of the 19 classes distribution revealed a consistent number of classes with an insignificant number of paragraphs associated: this led to the decision of deleting all the paragraphs where the total occurrences number of its label inside the corpus resulted to be less than 100.
This process lastly produced a dataset of 1878 rows and 7 definitive persuasion techniques:

Features	*Data type*	*Description*	*Value ranges*
Article	Int	Article identification number.	111111111 - 999001970
Paragraph	Int	Article paragraph identification number.	1 - 122

Technique	string	Persuasion technique assigned to a paragraph .	7 unique labels: Loaded_Language, Name_Calling-Labeling, Repetition, Doubt, Appeal_to_Fear-Prejudice, Exaggeration-Minimisation, Flag_Waving
ID	Int	Record identification number.	0 - 1877
Text	string	Paragraph textual content.	1878 unique texts

Tab 1. Dataset features descriptions.

3. Text Exploration

Before proceeding with the multi-class classification tasks, it was of our interest to investigate the contents and nature of the newspaper paragraphs texts. It has been chosen to focus on information like most frequent words in persuasion techniques, most recurrent named entities in political speeches and newspaper articles containing manipulation techniques, more correlated sentiment with texts presenting persuasion techniques and within each of the different techniques, etc.
In order to answer these questions, it was decided to proceed with a content word analysis, named entity recognition (NER) and sentiment analysis using the Python library SpaCy. In each of these analysis, information from all the paragraphs were extracted and analyzed considering their persuasion techniques separately in order to understand if some particular textual features were more correlated to specific persuasion techniques.

3.1 Content Word Analysis

Regarding content word analysis, a content word is defined as a part of speech that contributes in expressing the meaning of a text or speech. In linguistic, content words are considered an open class: new content words could constantly be added to a natural language through time, while some of them can leave as they become obsolete. Those features distinguish content words from function words, which are, instead, a closed class and do not provide any information regarding the meaning of a text since their function in a natural language is connecting and establishing relationships between content words or sentences (no new function word can be added to a natural language).
In our analysis, it was decided to consider nouns, verbs and adjectives as content words because those are the parts of speech prototypically associated with the meaning of texts and speeches. For each of the persuasion techniques different informations were extracted: the count of the total content words within the paragraph, the total and percentage of unique content words, the total and percentage of names, verbs and adjectives (and unique ones) separately. The overall results, considering all the paragraphs, are presented in the table below:

	Total	Percentage in texts	Total Unique	Perc. Unique
All Content Words	34960	33.4%	6745	19.3%
Nouns	16940	16.2%	3466	20.5%
Verbs	11234	10.7%	1643	14.6%
Adjectives	6786	6.5%	1636	24.1%

Tab 2. Content words informations extracted for each persuasion technique.

Observing the collected data, it is possible to ascertain that 33.4% of the tokens in the paragraphs consists of content words; also, *nouns* is the most common type of content words, followed by *verbs* and *adjectives*. This isn't a surprising fact: *adjectives* are optional and less frequent content words in

texts and speeches, while *nouns* and *verbs* are mandatory in syntactic composition. We can also observe that *verbs*, while being common in texts, have less variety in terms of uniqueness, with only 14.6% of unique verbs. *Adjectives* instead, while being less common, have the most variety with 24.1% of unique adjectives. In order to comprehend those facts, it was decided to extract the most common content words, shown in Table 3.

It is noticeable that the most common content words in the texts are verbs, like *say, have, make, tell, know*. This could be the reason why verbs have less variety. Also, those kind of verbs are very common in texts and related to human agents, so it would be expected to find a considerable number of people and organizations in the following NER. Regarding nouns and adjectives, we can affirm that they do have more variety and some of them are related to political topics and discussions. Similar distributions of content words were found for each distinct persuasion technique.

Most Common Content Word Format: ('entity', category), frequency)	Most Common Nouns Format: ('entity', frequency)	Most Common Verbs Format: ('entity', frequency)	Most Common Adjectives Format: ('entity', frequency)
('say', 'VERB'), 506)	('year', 170)	('say', 506)	('other', 125)
('have', 'VERB'), 301)	('people', 155)	('have', 301)	('many', 95)
('make', 'VERB'), 185)	('time', 138)	('make', 185)	('american', 88)
(('do', 'VERB'), 180)	('man', 118)	('do', 180)	('former', 86)
('year', 'NOUN'), 170)	('case', 96)	('go', 173)	('more', 83)
(('take', 'VERB'), 161)	('report', 94)	('take', 161)	('new', 80)
(('people', 'NOUN'), 155)	('fact', 89)	('know', 145)	('last', 76)
(('know', 'VERB'), 145)	('law', 80)	('tell', 129)	('public', 75)
(('time', 'NOUN'), 138)	('day', 77)	('be', 127)	('nuclear', 56)
(('tell', 'VERB'), 129)	('investigation', 75)	('come', 120)	('own', 54)

Tab 3. Most common content words extracted.

3.2 Named Entity Recognition

	Total	Percentage	Unique	Perc. Unique
All Entities	8078	100%	3390	41.97%
CARDINAL	457	5.65%	185	40.48%
DATE	907	11.23%	504	55.57%
EVENT	38	0.47%	29	76.32%
FAC	61	0.76%	22	36.07%
GPE	1263	15.64%	318	25.18%
LANGUAGE	9	0.11%	2	22.22%
LAW	38	0.47%	27	71.05%
LOC	114	1.41%	46	40.36%
MONEY	39	0.48%	34	87.18%
NORP	826	10.23%	175	21.19%
ORDINAL	100	1.24%	15	15%
ORG	1978	24.49%	904	45.70%
PERCENT	51	0.63%	45	88.24%
PERSON	1941	24.03%	886	45.65%
PRODUCT	51	0.63%	32	62.75%
QUANTITY	7	0.08%	7	100%
TIME	96	1.19%	69	71.86%
WORK OF ART	102	1.26%	90	88.24%

NER consists in extracting named entities from a dataset and classifying them in predefined categories such as *person, location, organization, monetary value*, etc.

Due to the fact that persuasion techniques and political speeches in general involve people and organizations, the expectation is to find a much higher number of entities belonging to the categories *PERSON*, *ORG* and *NORP* (a.k.a. nationalities, religious or political groups) compared to the other possible labels. From every paragraph, different kind of entities have been extracted and presented in table 4. From the data extracted it is ascertainable that the percentage of unique named entities in the texts is very high, settling at 41.97%.

Tab 4. Different kind of entities extracted for each paragraph.

This is also confirmed by the fact that, even within the most populated categories, the percentage of unique entities tends to be high. Regarding the different categories, as we expected, *PERSON* and *ORG* are the most common entities extracted by a large margin, with *1941* and *1978* occurrences respectively. Also, almost one half of the entities extracted belong to one of these categories. *GPE* (countries and cities), *DATE*, *NORP* and *CARDINAL* also have a fair amount of representation: this is also due to the fact that those kind of entities are recurrent in texts with political topics. *LANGUAGE* and *QUANTITY* are by far the least represented categories, with only 9 and 7 occurrences respectively. In order to have a better understanding of the distribution of the named entities, we extracted the most common one by paragraphs table 5. By looking at these entities, we can affirm that the most common entities mainly belong to categories described above; also, we can observe that the newspaper paragraphs are fairly recent and most of them talk about the American political scene. Similar distributions of named entities were found within each different persuasion technique.

Most common entities

Entities category	*Entities*	*Frequencies*
GPE	Iran	116
	Us	82
	Syria	45
	Russia	39
	America	36
ORG	Trump	104
	Church	99
	FBI	84
	CIA	37
	Guardian	31
	Assange	28
CARDINAL	one	85
	two	59
ORDINAL	first	56
DATE	today	33
NORP	American	77
	Democrats	40
	Muslim	39
	Islamic	37
	Catholic	33
	Russian	31
PERSON	Trump	72
	Francis	72
	Pope	38
	Clinton	33

Tab 5. Most common named entities and their distribution.

3.3 Sentiment Analysis

	Positive Sentiment	Negative Sentiment
Total	66.2%	33.8%
Loaded Language	65.5%	34.5%
Name Calling	70.9%	29.1%
Repetition	66.7%	33.3%
Prejudice	63.7%	36.3%
Exaggeration-Minimisation	66.6%	33.4%
Flag Waving	67.4%	32.6%

Tab 6. Mean of positive and negative sentiment for each paragraph.

For the sentiment analysis SpaCy was employed because of its ability in providing both positive and negative sentiment associated to each sentence or paragraph with percentages instead of binary labels. For each paragraph belonging to a particular persuasion technique, it was decided to calculate the mean of the positive and negative sentiment for each paragraph in order to provide a general overview of the sentiment associated with that particular persuasion technique. The starting hypothesis was that a paragraph containing a manipulation technique would carry a positive sentiment score in order to be effective and believable to a potential human reader, so the expected result was to find, for each percentage means associated to a paragraph, a score superior to 0.5 for the positive sentiment, regardless of the specific kind of persuasion technique. The obtained results are presented in table 6: the positive sentiment associated with each persuasion technique ranges from a minimum of 62.1% for the label *Doubt* to a maximum of 70.9% for the label *Name Calling*, with a total positive sentiment equal to 66.2%. Those results seem to corroborate the hypothesis declared above: in fact, the sentiment is more positive than negative regardless of the

specific type of manipulation, confirming the idea that persuasion techniques should be associated with a positive sentiment in order to be effective to human readers.

4. Data Preparation

The next phase focused on the labels distribution exploration in order to provide a reasonable base for the following classification processes.
To solve the imbalance of the classes distribution it was decided to reduce the cardinality of the majority class through a random undersampling approach, using a specific sampling strategy in order to preserve as much data as possible. This resulted in a dataset containing 1422 paragraphs for 7 distinct balanced persuasion techniques.

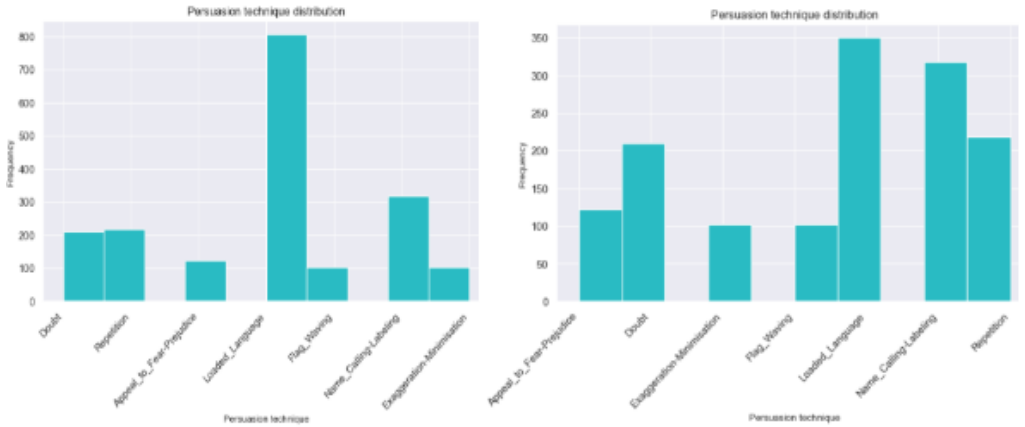

Fig 2. Previous and definitive dataset labels distribution.

Furthermore, the total number of lemmas and tokens (both distinct and not) was investigated, both without stopwords removal and with it:

Including stopwords		*Removing stopwords*	
Total token number (not distinct)	65968	*Total token number (not distinct)*	36890
Total token number (distinct)	8923	*Total token number (distinct)*	8787
Total lemmas number (not distinct)	65968	*Total lemmas number (not distinct)*	36890
Total lemmas number (distinct)	8095	*Total lemmas number (distinct)*	7962

Tab 7. Token and lemmas count.

5. Textual representation

To perform different classifications and determine which type of textual representation better fits the goal, it was decided to try different textual representation methods with different settings. Both Word2Vec and a CountVectorizer textual representation were employed for all the classification methods[1]. The specific chosen textual representation configuration depends on the classification methods[2]. The CountVectorizer method was applied on both tokenized and lemmatized text for the different implementations of SVM, Random Forest and Naive Bayes.
A first fine tuning for each classifier[3] was made considering the different parameters (which will be shown in their own sections) combined to these possible candidates for the feature creation and selection:

[1] Exceptions were made for Bert (which uses its own way of treating and analyzing texts) and LSTM, where only the CountVectorizer method was applied.
[2] Except for the CountVectorizer stopwords and accents removal, considered essential and necessary cleaning.
[3] Exceptions made for Bert and LSTM.

Pipeline elements	*Parameter*	*Function*	*Candidates*
Count Vectorizer	min_df	Threshold that allows to ignore terms that have a document frequency strictly lower than it.	3, 5, 7
	ngram_range	Lower boundary of the range of n-values for different word n-grams to be extracted.	(1,1), (1, 2), (1,3), (1,5)
SelectKBest	k	Number of top features to select	20, 50, 100, 300, 500, 700
TfidfTransformer	*norm*	Normalization type	l1, l2

Tab 8. Candidate values for feature extractor and selector.

It was decided to keep the parameter settings of the previously obtained classifiers and rather evaluate how changing the Word2Vec parameters impacted on the results in terms of accuracy because of:

1. computational limitations;
2. the important variety of parameter values that Word2Vec allows to set (which have been tested and illustrated in the *Word2Vec for classification* section);
3. results obtained through prior classifiers grid searches;

Also, rather than choosing the best configuration of Word2Vec depending on the single classifiers performances (that would result in creating different Word2Vec representations with different instantiations for each classifier) it was decided to choose the best Word2Vec parameter setting combination in terms of "compromise" between the different accuracy results of the 5 classifiers previously mentioned.

6. Classification

Different classifiers have been tested with the aim to obtain the best results in terms of accuracy. The chosen classifiers were:

- Linear Support Vector with One VS One strategy and One VS Rest strategy, Random Forest and Naive Bayes with both Word2Vec and a CountVectorizer[4] textual representation;
- Long short-term memory combined with CountVectorizer textual representation;
- Bert classifier.

6.1 Linear Support Vector Classifier

A Support Vector Machine is a linear model for classification problems that can solve linear problems by creating a hyperplane which separates the data into classes. The Linear version has the characteristic that it has greater flexibility in the choice of penalties and loss functions and those features have been exploited in the parameter tuning phases.
A first phase of tuning was made exploiting a pipeline grid search wich automatically performs a 5-fold cross validation in order to carry out a tuning for each possible different parameter for the count vectorizer, TF-IDF, k number of features selection (as explained in the *Textual Representation* section), and the classifier. The chosen candidates for each parameter consist in a combination of the ones presented in the *Textual Representation* section and the ones presented in Table9.

Pipeline Element (Classifier)			
LinearSVC	penalty	Specifies the norm used in the penalization.	l1,l2
	loss	Specifies the loss function.	hinge, squared_hinge

[4] Combined with TF-IDF for the feature selection.

multi_class	Determines the multi-class strategy if y contains more than two classes.	ovr, crammer_singer

Tab 9. Candidate values for different classifier parameters.

From this first tuning phase the best values for each desired parameter were:

- for what the CountVect is concerned, the best values for the *min_df* and *ngram_range* were respectively 3 and (1,3);
- for what SelectBestK is concerned, the best value for the *k* resulted to be *700*;
- for what TF-IDF is concerned, the best value for the *norm* resulted to be *l1*;
- for what the classifier is concerned, the best values for the *loss*, *multi_class* and *penalty* were respectively *squared_hinge*, *ovr* and *l2.*

The *ovr* multi-class strategy automatically applies a one-vs-rest classification approach.
At this point a final tuning was made only concerning the CountVectorizer different params configuration set (*min_df, ngram_range* and *their combination)*, since it was interesting to investigate how much an increasing number of linguistic factors considered could impact the final classifier accuracy. The parameters values involved in this final pipeline were extracted from the previous grid and tuned with the possible CountVectorizer configurations:

- {min_df: 3};
- {ngram_range: (1,3)};
- {min_df: 3, ngram_range: (1,3)}.

The best one resulted in {min_df': 3} with an accuracy of 0.34 on the validation set. This lastly has been implemented in the final classifier, which on the test set returned an accuracy of 0.37. The classifier has also been implemented using the Wordnet lemmatizer as CountVectorizer tokenizer method and leaving the other pipeline components parameters unchanged. This process returned an accuracy of 0.39, proving to be better than the default tokenization version.

6.1.1 One VS One strategy

Another implementation of the linear support vector classifier was made exploiting the one-vs-one approach, which splits the dataset into one dataset for each class versus every other class, actually converting a multiclass problem into a binary classification problem. Since a tuning of the SVM model has already been done for the base classifier implementation, it was decided to repropose the same classifier parameter values and to perform a pipeline grid search for the feature selection and generation parameter values, resulting in:

- for what the CountVect is concerned, the best values for the *min_df* and *ngram_range* were respectively 3 and (1,3);
- for what SelectBestK is concerned, the best value for the *k* resulted to be *700*;
- for what TF-IDF is concerned, the best value for the *norm* resulted to be *l1*.

After this process and also in this case, different CountVectorizer configurations attempts were made in order to investigate how much the increasing number of linguistic factors could impact on the final classifier accuracy:

- {min_df: 3};
- {ngram_range: (1,3)};
- {min_df: 3, ngram_range: (1,3)}.

The best one resulted in {min_df': 3} with an accuracy of 0.32 on the validation set.
This lastly has been implemented in the final classifier, which on the test set returned an accuracy of 0.32. The classifier has also been implemented using the Wordnet lemmatizer as CountVectorizer tokenizer method and leaving the other pipeline components parameters unchanged. This process returned an accuracy of 0.39, proving to be better than the default tokenization version.

6.2 Random Forest Classifier

Random forest is an ensemble learning method for classification that operates by constructing a multitude of decision trees at training and returns as output the class selected by most trees. Also for this classifier a first tuning phase was made exploiting a pipeline grid search in order to carry out a tuning for each possible different parameter for the count vectorizer, TF-IDF, k number of features selection (as explained in the *Textual Representation* section), and the classifier. The chosen candidates for each parameter are a combination of the ones presented in the *Textual Representation* section and the ones shown in Table 10.

Pipeline Element (Classifier)			
Random Forest	n_estimators	The number of trees in the forest.	100,150
	min_samples_split	The minimum number of samples required to split an internal node.	2,7
	criterion	The function to measure the quality of a split.	gini, entropy, log_loss
	max_features	The number of features to consider when looking for the best split.	sqrt, log2, None
	min_samples_leaf	The minimum number of samples required to be at a leaf node.	2,7

Tab 10. Candidate values for different classifier parameters.

From this first tuning phase the best values for each desired parameter were:

- for what the CountVect is concerned, the best values for the *min_df* and *ngram_range* were respectively 3 and (1,3);
- for what SelectBestK is concerned, the best value for the *k* resulted to be *500*;
- for what TF-IDF is concerned, the best value for the *norm* resulted to be *l1*;
- for what the classifier is concerned, the best values for the *criterion*, *max_features* *min_samples_leaf, min_samples_split* and *n_estimators* were respectively *gini*, *log2, 2, 2* and *150.*

At this point a final tuning was made only concerning the CountVectorizer params different configuration set, considering all the possible combinations of parameters (*min_df, ngram_range* and *their combination)*, since it was interesting to investigate how much the increasing number of linguistic factors to consider could impact on the final classifier accuracy. The values for all the parameters involved in this final pipeline grid search were extracted from the previous grid and for the different CountVectorizer configurations:

- {min_df: 3};
- {ngram_range: (1,3)};
- {min_df: 3, ngram_range: (1,3)}.

The best one resulted in {min_df': 3} with an accuracy of 0.31 on the validation set. This lastly has been implemented in the final classifier, which on the test set returned an accuracy of 0.35. The classifier has also been implemented using the Wordnet lemmatizer as CountVectorizer tokenizer method and leaving the other pipeline components parameters unchanged. This process returned an accuracy of 0.39, proving to be better than the default tokenization version.

6.3 Naive Bayes Classifier

The Naive Bayes classifier belongs to the family of simple "probabilistic classifiers" based on applying Bayes theorem with strong independence assumptions between the features. A first phase of tuning was made exploiting a pipeline grid search in order to carry out a tuning for each possible different parameter for the count vectorizer, TF-IDF, selection of k number of features (as explained in the *Textual Representation* section), and the classifier in order to get one optimal value for each

parameter. The chosen candidates for each parameter are the ones presented in the *Textual Representation* section and the ones presented in Table 11.

Pipeline Element (Classifier)			
Naive Bayes	var_smoothing	Portion of the largest variance of all features that is added to variances for calculation stability.	1e-11, 1e-10, 1e-9
	priors	Prior probabilities of the classes.	None

Tab 11. Candidate values for different classifier parameters.

From this first tuning phase the best accuracy value (0.31) was reached for this parameter setting:

- for what the CountVect is concerned, the best values for the *min_df* and *ngram_range* were respectively 3 and (1,1);
- for what SelectBestK is concerned, the best value for the *k* resulted to be *700*;
- for what TF-IDF is concerned, the best value for the norm resulted to be l1;
- for what the classifier is concerned, the best values for *priors* and *var_smoothing* were respectively *None* and *1e-10*.

Because of the value (1,1) returned by the grid search for the *ngram-range* parameter, which coincides with its default value, there was no need to run an additional pipeline grid search to evaluate different settings and combinations of linguistic features.
Only considering the {min_df': 3} CountVectorizer optional parameter in the final classifier, the result consists of an accuracy of 0.29 on the test set. The classifier has also been implemented using the Wordnet lemmatizer as CountVectorizer tokenizer method and leaving the other pipeline components parameters unchanged. This process returned an accuracy of 0.31, proving to be better than the default tokenization version.

6.4 Word2Vec for classification

As regards the performance of the models trained using the textual representation provided by Word2Vec (in its CBOW variant) it was decided to use as classifiers parameter values those used in the final training of the previous models so that the performances would be comparable with CounterVectorizer results. To choose the best values for the Word2Vec parameters it was decided to try all the possible value combinations exploiting a nested for loop performed on a validation set, composed of 10% of the training set. The parameters values explored in combinations are:

- vector_size = [100,300,500,700];
- window = [2,5,7];
- min_count = [2,5].

The choice fell on the set of parameters that returned the best results in terms of accuracy on as many classifiers as possible. The best parameter set turned out to be 500 for *vector_size,* 7 for *window,* 2 for *min_count,* which returned the following accuracy values on the validation set:

- the Random Forest Classifier, SVC classifier and it's different strategy implementation got an accuracy of 0.25;
- Naive Bayes classifier got an accuracy of 0.13.

The results in terms of accuracy of the various models on the test set were 0.25 on all the classifiers considered until now except for Naive Bayes, which returned an accuracy of 0.10.

6.5 LSTM Classifier

It was decided to try the LSTM approach because paragraphs can also be composed of several sentences and it was interesting to evaluate how much the different components of a paragraph could result helpful for the classification of persuasion techniques.
In order to proceed with the classification via LSTM it was necessary to carry out a one-hot encoding on the 7 possible labels so that the labels dimensionality would be equivalent to the number of the model final layers. The textual data were represented through CountVectorizer with the deemed necessary cleaning parameters (accents and stopwords removal) and a minimum frequency of 3 inside the corpus, since it was the best-resulting setting from the previous classifications. The model is composed of three layers:

- the *Embedding layer*: enables to convert each word representation into a fixed length vector of defined size; it has three parameters that allow us to define the vocabulary size, the words vector length and the maximum sequence length;
- the *LSTM layer*: it has been implemented with 3 parameters, which respectively define the number of how many LSTMs to stack on top of each other, the fraction of the units to drop for the linear transformation of the inputs and the fraction of the units to drop for the linear transformation of the recurrent state;
- the *Dense layer*: this layer is deeply connected to every neuron of its preceding layer. The dense layer is asked to provide the output array of shape 7 using the *softmax* activation function since the labels are one-hot encoded and the model loss function is the *categorical_crossentropy*.

Finally the model was fitted using a validation portion of 10% of the whole training, a batch size of 100 and an early-stopping callback, which stops training when the loss on the validation has stopped improving since going forward would result in overfitting.
After different trials focused on changing parameters as the epochs, the optimizers (SGD, Adagrad and RMSprop), number of LSTM hidden layers, and the other different parameters illustrated inside the layers presentation, the best parameter setting in terms of accuracy result to be the one definitively presented within the notebooks, which returned a validation accuracy of 0.26 in the last performed epoch on the validation set and a final accuracy of 0.25 on the test set.

6.6 BERT Sequence Classifier

Finally, the Bert for Sequence classifier was implemented in order to evaluate whether a transformers language model could return better results than the previous classifiers, proving to be suitable for the task objective.
In addition to the preprocessing exposed in the data understanding and preparation section, in order to implement BERT was necessary to carry out some fundamental steps required by the classifier:

- perform a mapping of the labels to result in integers rather than strings;
- format the sequences so that they could be processed by BERT, therefore inserting specific tokens (CLS at the beginning of the sequence and SEP at the end) and defining a maximum length as a limit of the *case-unfold* model.

After realizing a validation set to test different parameter settings and following the necessary procedure to make the textual data subjectable to BERT (index the elements in the sequences, convert each input data format into torch tensor type, etc.) the pretrained bert model *bert-base uncased* was loaded through the *BertForSequenceClassification* transformer class and different weights combinations were set-up.

Whereupon, optimizer and scheduler with default values setting were instantiated.
After the definition of the trainer and evaluation methods, different parameters combination trials were made; some attempts results (in terms of accuracy for brevity) are shown in Table 12.

Parameters combinations	*Validation Accuracy*	*Test Accuracy*
batch_size = 8, weight_decay = 0.01, learning_rate= 2e-5, epochs = 4, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.41	0.35
batch_size = 32, weight_decay = 0.01, learning_rate= 2e-5, epochs =4, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.38	0.29
batch_size = 64, weight_decay = 0.01, learning_rate= 2e-5, epochs =5, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.30	0.28
batch_size = 8, weight_decay = 0.1, learning_rate= 2e-5, epochs = 6, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.47	0.40
batch_size = 8, weight_decay = 0.01, learning_rate= 5e-5, epochs = 3, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.41	0.35
batch_size = 8, weight_decay = 0.01, learning_rate= 5e-5, epochs = 5, eps_set=1e-8, no_decay = ['bias', 'LayerNorm.weight','gamma', 'beta']	0.54	0.51

The best combinations of parameters values resulted to be the last of the previously exposed ones, reaching the best accuracy level, even if returning different flaws in terms of precision and other evaluation metrics.

Tab 12. Classifier performances for different parameter settings.

7. Conclusions and Final Considerations

The results of the tested classifiers have proved to not be up to this experimental classification task, despite the different possible fine tuning attempts and the various textual representation approaches which have been explored.
Regarding Linear Support Vector classifier and its different approach, Naive Bayes and Random Forest classifier, the best results in terms of accuracy have been reached through the employment of the CountVectorizer textual representation rather then Word2Vec embedding, which returned unexpectedly low results where employed. Probably, using different embedding parameters settings for each single classifier would have accomplished better results. However, this alternative has not been pursued because the predictions shown for the different combinations of parameter values did not return significantly improved results, therefore the computational effort of carrying out different embeddings has not been valued worthy in terms of results.
For what the Linear Support Vector classifier and its different approach is concerned, better classification performances were achieved through the *lemmatizer tokenizer method* rather than the default tokenizer for the CountVectorizer. This behavior was expected from the moment that linear classifiers can make better distinctions if it considers standardized data rather than data with high variance (typical of language elements).
The worst employed classifiers reasonably resulted to be the LSTM classifier with its 0.25 accuracy score on the test set and the Naive Bayes, which returned an accuracy of 0.29 for the default tokenizer version, 0.31 with the lemmatizer version implemented in CountVectorizer and 0.10 for the Word2Vec textual representation implementation.
The best classifiers (excluding BERT) resulted to be the Random Forest, Linear Support Vector and its *One Vs One* implementation approach, which all reached an accuracy of 0.39 for the *lemmatizer* version of the CountVectorizer textual representation.
Finally, BERT was the only classifier to be able to reach the 0.5 accuracy threshold, making it the best one among all the others employed, coherent with the expectations.
Despite this, unfortunately, none of the experiments has led to significant results and this may be due to several unfavorable factors that accompanied the project: the number of samples were much lower than what would normally be required to satisfactorily train machine learning models, also considering the consistent number of investigated classes. Even without filtering (necessary in order to create the desired multiclass task) the number of actually annotated (multi-labeled) samples corresponded to 3760 paragraphs for 23 different labels, that could be still considered a low number of samples to address an experimental task which tries to predict factors that are still difficult to

define in psychological terms. Moreover, the implementation of *data augmentation* on such a small set of data could certainly have preserved the statistical properties of the data provided, but the provided data would still be so limited that projecting these statistical properties could have led to not accurate assumptions and conclusions.

Finally, another potential problem could reside in the way the data has been annotated: despite the efforts of "schematizating" the annotation decision process, the task nature itself still leaves a lot of space for the interpretation and subjectivity of the annotators, which is mostly an inevitable factor, considering the low objectivity of the task. In the end, since the result on such a small data sample was still sufficient and shows a margin of generalization by the models, if a large portion of data were provided there could be a good chance of being able to obtain classifiers capable of achieving acceptable performance.

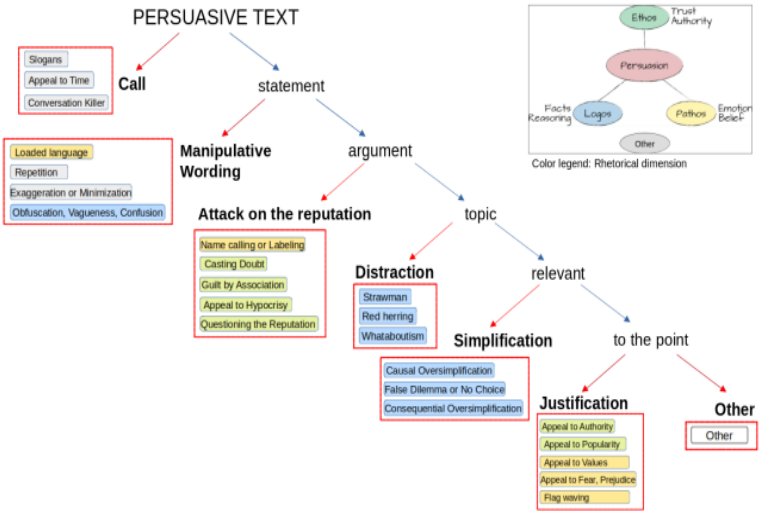

Fig 3. Decision diagram to determine which high level approach is used in a text (extracted from the annotation guidelines draft).[5]

[5] https://propaganda.math.unipd.it/semeval2023task3/data/annotation_guidelines.pdf