# Leveraging Pre-Trained BERT Model from *huggingface* to find stakeholder groups in argument data and comparing performances with Support Vector Machine Implementation.

Lilia Grasso

Statistical Natural Language Processing

Bielefeld Universität

Mat. 4161421

**Keywords:** debates, classification, imbalanced data, users groups, SVM, Natural Language Processing, transformers, huggingface, fine tuning pre-trained model, BERT for sequence classification

## ABSTRACT

*In 2022, a lot of people have an internet account and actively join to discussions in social networks, share posts and give information about personal details. Is it possible to understand their characteristics by just reading the comments and arguments they wrote on social media? To answer this question, Natural language processing tools can be used, tools that drive force behind machine intelligence in many modern real-world applications, as text classification. This paper analyzes the performances and the leveraging of the BERT Model and the Support Vector Machine model when classifying users, according to debates where they participate and their stances on them on the website Debate.org, into groups denoting some sensitive data, as the religion beliefs, the ethnicity and the political ideology. The BERT model is a pre-trained transformers from the huggingface library and in order to fine-tune it, some investigations on the tokenizer and the trainer methods have been carried out, to optimize the performance metrics, such as F1 scores. First of all, this model is implemented for predicting one label at time, and then improved to predict all labels together. The performance metrics have been compared with the resulting ones from the Support Vector Machine learning model for classification tasks. Results showed that when a class imbalance data is fed to the single label BERT trainer, the performances are poor compared to the ones of the SVM, especially when trying to understand religion beliefs and political ideas. Also, more words and sentences are used to train the SVM, higher become the metrics scores, especially when, in SVM, the imbalance of the classes is taken into account and solved. On the other hand, the scores of the multi label BERT are higher than both SVM and single label BERT, resulting in a powerful model that could predict, for example, Gender, with a F1 score of 0.82.*

## CONTENTS

## 1 INTRODUCTION

”The simplicities of natural laws arise through the complexities of the language we use for their expression”

*Eugene Wigner*

The field of Natural language processing (-i.e. NLP) is the branch of computer science -and more specifically, the field of artificial intelligence or AI— that tries to give to the computer the ability to understand text and words in the same way humans do [1]. A pretty complex task, because the human mind is a huge Neural Network that still has its own mysteries. NLP combines deep learning models, machine learning, and computational linguistics—rule-based modeling of human language—with statistical methods. Using these technologies, it’s possible to enable computers to process human language in the form of text or voice data and to ”understand” its full meaning, with the writer’s intent and sentiment too. NLP drives computer programs that translate text from one language to another, or that summarize large volumes of text rapidly—even in real time. As a matter of fact, Natural language processing is the driving force behind machine intelligence in many modern real-world applications, as for example, spam recognition, Virtual agents and chatbots, Social media sentiment analysis or Text summarization. But is it possible to use this fields to try to profile a person? On the internet almost everyone has an account on some social network, writing down comments or participating in discussions, debating and explaining their point of view to win some arguments. Could those texts be used to understand who they are, in what they believe, what is their income, or even what is their home country? This tasks can be seen by the Statistical Natural Language Processing as a Classification tasks, trying to understand from lines and lines of texts in which category a person belongs to. In order to achieve this goal, many tools can be used. For example, the Python programming language has the Natural Language Toolkit, or NLTK, an open source collection of libraries, programs, and education resources for building NLP programs.This includes methods also for subtasks, such as word segmentation, sentence parsing or word stemming and lemmatization (methods of trimming words down to their roots), and tokenization (for breaking phrases, sentences, paragraphs and passages into tokens that help the computer better understand the text). Or more advanced tools have been created such as transformers, a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease such as *huggingface* [2]. In this paper both of these ways have been discussed in order to find the best one to find stakeholder groups in argument data.

## 2 AIM OF THE ANALYSIS

The goal of the project is to find a model for classifying a user from the website DebateOrg into a group, denoting his/her characteristics, analyzing the debates' title in which he/she participated, the side (pro or against) and finally, the text that he/she wrote in those debates. More specifically the class into which the model should classify the user are:

- Religion: Non Believer, Christian, Muslim, Other;
- Gender: Male, Female, LGBTQIAPK+;
- Ideology: Left, Center, Right, Other;
- Ethnicity: Asian, Black, East Indian, Latino, Middle Eastern, Native American, Other, Pacific Islander, White.

In order to find the right model, the analysis is performed on two parallel ways: the first part involved the implementing of a Support Vector Machine to classify users' text and title's debates in the group above, while the second part is to implement and to leverage a pre-trained BERT model from Hugging Face Library[2] to do the same. BERT stands for Bidirectional Encoder Representations from Transformers [3], and its architecture consists of several Transformer encoders stacked together where each Transformer encoder encapsulates two sub-layers: a self-attention layer and a feed-forward layer. BERT is a powerful language model because is pre-trained on unlabeled data extracted from BooksCorpus, which has 800M words, and from Wikipedia, which has 2,500M words, and also it learns, at the same time, information from a sequence of words not only from left to right, but also from right to left.

## 3 DATA SET DESCRIPTION

The data set used was already built through a crawling from the website DebateOrg and it consists in 6 tables:

- debates: contains information about the debate;
- arguments: contains the arguments of the debates;
- users: contains the profile information;
- big_issues: contains the "big issue" vectors of the individual users;
- comments: still empty, should actually contain the comments on the debates;
- votes: still empty, should actually contain the votes for the debates.

The most important tables for the analysis are the users table, the debates table and the arguments table. They have been extensively pre-processed to get the perfect data set to feed in both of the models. The debates table contain data on each debate, providing the following important information: the index of the debate, the title, the username of the debate's creator, the side of the creator, the debate's category, the username of the second participant and his side. On the other hand, linked to the index of the debate, in the arguments table there are the texts written by the users per each discussion. The last, but not the least, table is the user table which provides a great amount of information on the people who are participating in the debate: for which party they vote, in which religion they believe, which is their gender or ethnicity, if they're in a relationship or not, how much they earn, what is their win ratio in the debates and so on. These can all be used as target variables in further developments of the models, but for the scope of this report, just religion, gender, ethnicity, and ideology are used. This choice was made because, in order to identify a person overall, it is not user's income or the school that users attended or if the users are in a relationship or not that shape their personalities or their being, but it's in what they believe and which is their country and their gender that matters. Thanks to such information is it possible to understand the environment they grew up and to really profile someone just reading at what they're writing.

## 4 DATA CLEANING

In order to train the models and to implement them in the best way possible, the most important part is to clean and pre-process data, to obtain a valid data set. To achieve this goal, just few columns from the users and debates tables were selected: username, title, category, user's side, from the debates table; user, ideology, gender, ethnicity, religion from the users table; later on also the comments in argument table are used. Furthermore, from the users characteristics, the class "Not Saying" it has been removed, because it didn't add any valuable information to the data. Considering that per each category the classes were more than 5, they have been grouped in some macro-categories as shown in figure 1.

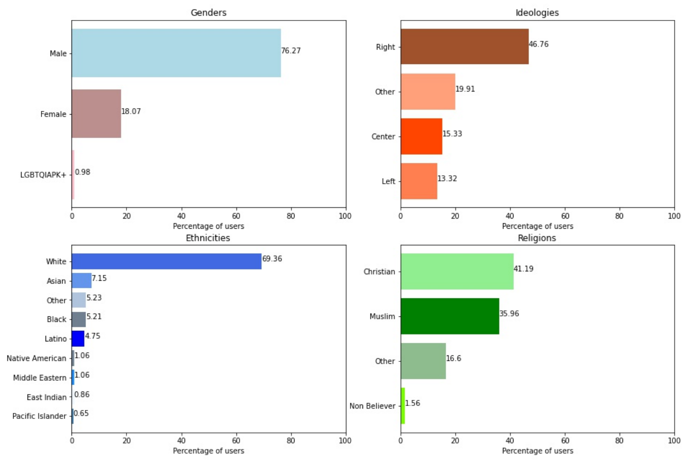
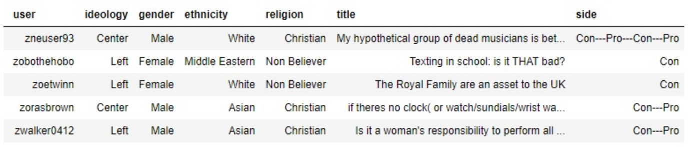


**Figure 1.** Bar plots of users per each categories

The noticeable takeaway from the plots is that all the groups have imbalanced classes, where the predominant characteristics of the users are White, Christian, Rightist, Male. This fact needs to be taken into account because it can affect the performance of the models.

Moving on, the second step has been to left-join arguments table with the debates data on the debates' index variable to obtain a table where to each arguments written by a user, a title is associated. This data set has been left-joined with the user data set, on the username variable, in order to have per every person, his information on the groups in which he belong. The third, and last step has lead to the creation of two main data set: it has been possible to create the first data set grouping once by user, user's side and debate's title, combining all the text arguments per each debate, and the second time just by user, combining the debates' titles and users' side and the further arguments, leading to have data set where each row corresponds to one users, as shown in figure 2.



| user | ideology | gender | ethnicity | religion | title | side |
|---|---|---|---|---|---|---|
| zneuser93 | Center | Male | White | Christian | My hypothetical group of dead musicians is bet... | Con---Pro---Con---Pro |
| zobothehobo | Left | Female | Middle Eastern | Non Believer | Texting in school: is it THAT bad? | Con |
| zoetwinn | Left | Female | White | Non Believer | The Royal Family are an asset to the UK | Con |
| zorasbrown | Center | Male | Asian | Christian | if theres no clock( or watch/sundials/wrist wa... | Con---Pro |
| zwalker0412 | Left | Male | Asian | Christian | Is it a woman's responsibility to perform all ... | Con---Pro |

**Figure 2.** Example of the data set.

## 5 METHODOLOGICAL ASPECTS

### 5.1 Support Vector Machine Implementation

The Support Vector machine [4] are a set of supervised learning methods used mainly for classification, but can be used also for regression and outliers detection. The main advantages of support vector machines are that they are effective in high dimensional spaces and still effective in cases where number of dimensions is greater than the number of samples. Also, SVM uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. Furthermore, different Kernel functions can be specified for the decision function [5]. One of the goal of this project is to implement a MultiOutput Support Vector Machine, capable of classifying the users into the labels of he groups Religion, Ideology, Gender and Ethnicity, using the debates' title and users' sides.

*5.1.1 Data Pre-Processing* The Pre-processing part is an important step in any data mining process. This involves transforming raw data into an understandable format for NLP models, -i.e. making them "machine-readable". Data pre-processing is a method of resolving the incompleteness, inconsistencies of real-world data that are also likely to contain many errors. The techniques used in data pre-processing are the following:

- Tokenization: This is a process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing. NLTK Library has *word_tokenize* and *sent_tokenize* to easily break a stream of text into a list of words or sentences, respectively.
- Word Stemming/Lemmatization: they are both used to reduce the inflectional forms of each word into a common base or root. Lemmatization is closely related to stemming, with the main difference that, a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

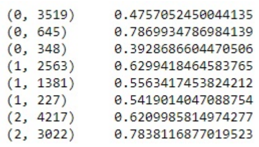
The order of the operations applied on the debates' title column (per users) is the following:

- Remove Blank rows in the text, if any;
- Change all the words to lower case;
- Word Tokenization;
- Remove Stop words;
- Word Lemmatization.

The output will be (directly taken from one of the user's debates titles):

| Raw Text | Pre-processed Text |
|---|---|
| "America is the best nation.— If God farted, would it smell beautiful or disgusting?— Should the Board of Education include the Bible into the Public School Curriculum?" | "['america', 'best', 'god', 'fart', 'would', 'smell', 'beautiful', 'disgust', 'board', 'education', 'include', 'bible', 'public', 'school', 'curriculum']" |

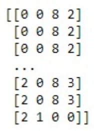
The data set will be split into 3 sets: 80% for the train data set, 10% for the evaluation data set and 10% for the test data set. The X variable is a data set containing the titles of debate where each user participated and the side per each title, while the Y variable is a data set containing the labels of gender, ideology, religion and ethnicity. In order to train the SVM, the pre-processed text is still not ready, it needs to become "machine-readable". To achieve this task, the pre-processed text is vectorized: this is a general process of turning a collection of text documents into numerical feature vectors. To convert text TF-IDF (hacronym than stands for "Term Frequency — Inverse Document" Frequency i.e. components of the resulting scores assigned to each word) has been used. The Term Frequency summarizes how often a given word appears within a document. The Inverse Document Frequency down scales words that appear a lot across documents. Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents. This will help TF-IDF build a vocabulary of words which it has learned from the corpus data and it will assign a unique integer number to each of these words. The vectorized data will contain for each row a list of unique integer number and its associated importance as calculated by TF-IDF, as shown in the vectorized data set in the figure 3. In order to add the sides of the users, after encoding it with LabelEncoder (assigning 0 to CON and 1 to PRO), the vectorized X has been concatenated with the Encoded side -i.e. it has been stacked horizontally (column wise), using *hstack* method from library *scipy.sparse*.



```
(0, 3519)     0.4757052450044135
(0, 645)      0.7869934786984139
(0, 348)      0.3928686604470506
(1, 2563)     0.6299418464583765
(1, 1381)     0.5563417453824212
(1, 227)      0.5419014047088754
(2, 4217)     0.6209985814974277
(2, 3022)     0.7838116877019523
```

**Figure 3.** First number is the row number of "Train_X_Tfidf"; second number is the unique Integer number of each word in the first row and the third is the score calculated by TF-IDF Vectorizer

The Y variable needs to be "machine-readable" too, and this task has been completed by encoding, then applying the method *fit_transform* to every group of labels separately, and then stacking them together, in order to obtain a vector for Y train and then a vector for the Y test. In figure 4 is shown the Y train vector. The labels are given from 1 to the number of labels, in an alphabetical order, that is, for example for gender, 0 to Female, 1 to LGBTQIAPK+ and 2 to Male and so on. Considering the



```
[[0 0 8 2]
 [0 0 8 2]
 [0 0 8 2]
 ...
 [2 0 8 3]
 [2 0 8 3]
 [2 1 0 0]]
```

**Figure 4.** First column is the encoded gender, second is the religion, third is the ethnicity, fourth is the ideology

first row in figure 4, the first 0 is for Female, the second 0 is for Christian, the 8 is for White and the 2 for Other ideologies.

*5.1.2 Training* In order to train the Support Vector machine for classifying users from title's debates and and their sides, first a SVC (Support Vector Classification) method from *sklearn.svm* library has been initialized with the following parameters:

- kernel = "linear": it specifies that the kernel to be used in the algorithm is linear;
- class_weight = "balanced": it specifies the weights for each class. It has been chosen "balanced" because data have highly imbalanced classes and this mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as $\frac{n\_samples}{(n\_classes \cdot np.bincount(y))}$;
- gamma = "auto": it specifies which kernel coefficients to use, in this case it uses $\frac{1}{n\_features}$;
- probability = True: it enables probability estimates.

In order to fit the data, and to predict all the groups, it has been initialized a Multi Output Classifier, because there is not only one variable to predict, but four in total. The results and the evaluations, together with the comparison between SVM metrics and BERT metrics, will be discussed in the **Evaluation & Results** section.

## 5.2 Leveraging BERT Model

*5.2.1 Data Pre-Processing* This section is focused on training the BERT model with the data in figure 2, that has information on title and side of the user, and then to compare performances with SVM.

First of all, a train, test and evaluation data split has been applied, as in the SVM part: 80% of the values have been inserted in the train set, 10% in the evaluation set and 10% in the test set. BERT model expects a sequence of tokens (words) as an input. In each sequence of tokens, it will expect as an input: the token [CLS], that is the first token of every sentence and stands as a classification token; the token [SEP] that makes BERT know which token belongs to which sequence. If there is only one sequence, then this token will be appended to the end of the sequence. So, as a first step, sentences from the text need to be transformed as a sequence of tokens (tokenization part), and then, the tokens [CLS] and [SEP] needs to be added. This is done by using *BertTokenizer*, where in input there is the variable "title" paired with the "side" variable. The hyper-parameters were set as follows in train, evaluation and test data set:

- add_special_tokens=True,
- max_length=22,
- truncation="longest_first",
- padding="max_length",
- return_tensors="pt".

The parameter *padding* is set to "max_length" in order to pad each sequence to the maximum length that is, in this case, 22. This length is maximum length of each sequence. The *truncation* parameters is set to "longest_first", that means that the tokenizer will compare the length of both text and text_pair (side of the user) everytime a token needs to be removed and remove a token from the shortest. The last parameter is *return_tensors*, that it's the type of tensors that will be returned, and is set to "pt" because PyTorch is used. The reason why those kind of values for the parameters have been chosen is explained in the next subsection.

The output will be:

- input_ids: it is the id representation of each token;
- token_type_ids: it is a binary mask that identifies in which sequence a token belongs. If only one single sequence exists, then all of the token type ids will be 0.
- attention_mask: it is a binary mask that identifies whether a token is a real word or just padding. If the token contains [CLS], [SEP], or any real word, then the mask would be 1. Otherwise, if the token is just padding or [PAD], then the mask would be 0.

The readable data set, is not ready yet, because the labels are missing. In order to add them to create a tensor-machine-readable-data set, the batched X obtained from the BertTokenizer has been embedded to the labels variable separately, obtaining a tensor per each group. That means the a separated BERT model has been trained per each label (one for Religion, one for Gender, one for Ideology and one for Ethnicity).

*5.2.2 Training* In order to train the BERT for Sequence Classification, the trainer has been initialized with the following Training Arguments:

- num_train_epochs=3;
- learning_rate=1e-4;
- save_total_limit=1;
- evaluation_strategy = "epoch";
- save_strategy = "epoch";
- metric_for_best_model="eval_loss".

The parameter *num_train_epochs* is the total number of training epochs to perform. It has been chosen 3 because, if it was higher, the computation would have been very expensive. The parameter *learning_rate* (defaults to 5e-5) is the initial learning rate for AdamW optimizer, and it's set to 1e-4. Concerning *save_total_limit*, 1 is passed and that means it will limit the total amount of checkpoints to 1. The parameter *evaluation_strategy* refers to the strategy to adopt during training. In this case the value is *epoch*, that means the evaluation is done at the end of each epoch. While *save_strategy* is the checkpoint save strategy to adopt during training. In this case "epoch" has been chosen, meaning that the saving is done at the end of each epoch. In the last parameter *metric_for_best_model* has been chosen the metric to use to compare two different model, and in this case is *eval_loss*, which means that it will compute the loss on the evaluation data set per each epoch (strictly connected to *evaluation_strategy* and *save_strategy*). Futhermore, a train data set and an evaluation dataset have been given as an input in the training, with also a parameter set on *compute_metrics* in order to compute accuracy, f1 scores, precision and recall everytime a prediction is made. When the training is done, it usually makes prediction on the evaluation data set and in order to have a best overview on the metrics of the model, a CallBack has been added, in order to make prediction also on the Training data, computing the metrics. The results and the evaluations, together with the comparison between SVM metrics and BERT metrics, will be discussed in the **Evaluation & Results** section.

*5.2.3 Investigating on the best values for max_length and truncation* The parameters has been chosen after investigating on how the truncation and the padding of the Tokenizer work on the whole data set, applying the tokenizer to the debates' titles column paired with the users' side column. First of all, Tokenizer with no truncation and no padding has been initialized (truncation=False, padding=False) and per each sentence, the length of the list with tokens has been stored in a list. The mean of the list was computed and it was equal to 123.24. As the figure 5 shows, the lengths were very different and some of them were really high. Second of all, tokenization using truncation to max model input length and padding to max sequence in batch (truncation=True,

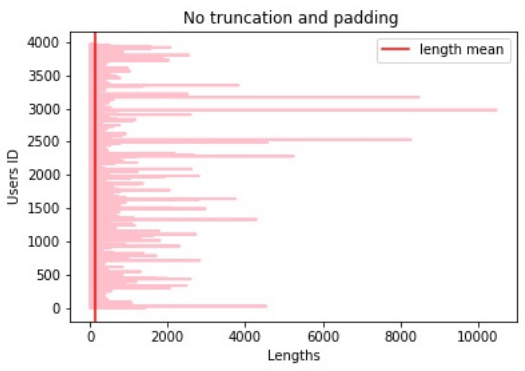


**Figure 5.** Lengths per each users

padding=True) is carried out and it resulted in a list of lengths that had mean equals to 87.38. As expected the lengths that were higher than 512 have been truncated to 512, while the remaining lengths kept their own, as shown in figure 6. Finally, tokenization using

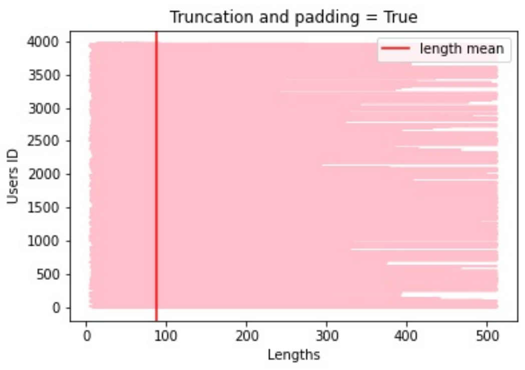


**Figure 6.** Lengths per each users with padding and truncation

truncation to max model input length and padding to max model input length (truncation=”longest_first”,padding=”max_length”) is carried out without specifying any max_length, resulting in a mean length equals to 512 because, if max_length is not stated, the tokenizer will suppose it’s equal to the maximum length accepted, -i.e. 512. That means that if the list of tokens is lower than 512, [PAD] tokens are added to reach this max number. As a quick recap, the results are:

1. Truncation=False, Padding=False $\rightarrow$ 123.24;
2. Truncation=True, Padding=True $\rightarrow$ 87.38;
3. Truncation=”longest_first”, Padding=”max_length” $\rightarrow$ 512;

Then, the question is, why 22? First, let’s look at the number of tokens in debates’ title in figure 7, computing by encoding to the max length 512 every title, to have an overview of the distributions if the length was maximal.

Most of the titles contain less than 300 tokens or more than 512. So, should the length be 512, or should be lower? In order to find an answer, several model have been trained, each with an input obtained by setting, in the tokenizer, different max lengths.This training has been carried out per each group of the target values and the results are similar to each other. For the sake of simplicity it has been decided to show the values just for F1 scores related to the model classifying the users in their different genders. For each length the F1 scores obtaining by predicting

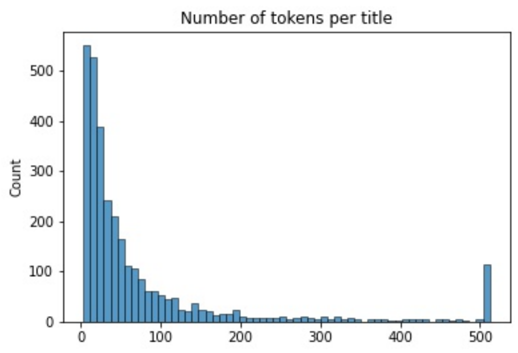


**Figure 7.** Distributions of number of tokens in title

the gender on the test set of the different models have been plotted in figure 8. The figure 8 clearly shows a constant trend from 22 to higher

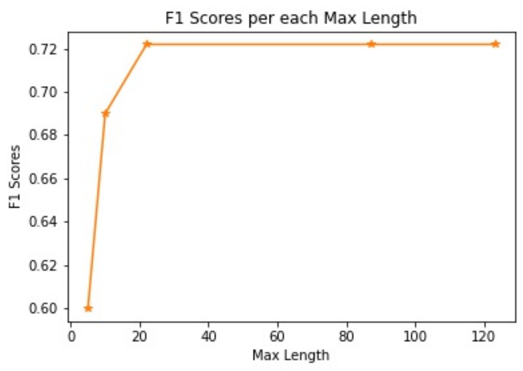


**Figure 8.** F1 Scores per each Max Length

values, while decreasing when lower lengths are used. In order to smooth the complexity and to choose the model with low computational expense, the max_length equals to 22 has been chosen to train all the models.

## 6 EVALUATION & RESULTS

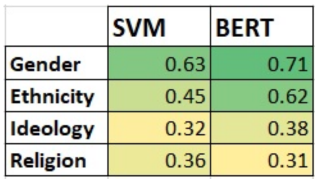
The metric used to compare the performances of the two models is the F1 score, that is one of the best metric to use when there is an imbalanced data, because it takes into account not only the number of prediction errors that your model makes, but it also looks at the type of errors that are made. The foundation of the F1 score are the *precision* and the *recall*:

$$Precision = \frac{TP}{TP+FP}; \quad Recall = \frac{TP}{TP+FN} \tag{1}$$

Where $TP$ is the number of True Positive, $FP$ is the number of False Positive and $FN$ is the numbe of False Negative. Ideally, a high precision and a high recall is ideal: a model that identifies all of the positive cases and at the same time it identifies only positive cases. But, unfortunately, in real life, Precision-Recall Trade-Off is something to deal with. It represents the fact that in many cases, it’s possible to tweak a model to increase precision at a cost of a lower recall, or to increase recall at the cost of lower precision. As mentioned above, they can be combined in one metric, the F1 score, using the harmonic mean, that is the correct mean to use when ratios are averaged:

$$F1score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2}$$

The maximum value reachable by the F1 score is 1, that means that if its value is closer to 1, both precision and recall are high, if it's closer to 0, both precision and recall are low and its value depends heavily on how imbalanced training data set is. The following values in figure 9 are the F1 scores of both the BERT model and the SVM computed by predicting the values of the test set and comparing the classifications, identifying the True positive or the False Positive etc.



| | SVM | BERT |
|---|---|---|
| Gender | 0.63 | 0.71 |
| Ethnicity | 0.45 | 0.62 |
| Ideology | 0.32 | 0.38 |
| Religion | 0.36 | 0.31 |

**Figure 9.** F1 scores computed on classifying debates' titles of the test set.

The values clearly shows some differences in both of the model. The first thing that can be pointed out is that the BERT classifier has higher performances than the SVM in classifying users into groups, at least for gender, ethnicity and ideology, especially the classification of ethnicity has a very high F1 Score, with $F1 = 0.62$, compared to $F1 = 0.45$ of the SVM. Furthermore, the values for Ideology and Religion are very low for both model and this can occur for many different reasons that depend on the domain, type system complexity, appropriateness of training documents and other factors. One reason could be that people have very subjective point of view, even if they belong to the same party or believe in the same religion. Trying to reduce the thousand different facets and faiths of the Muslim religion in a simple label that mentions "Muslim", or the dozens of parties that are leftist but with different ideas and enclose everything in a wing, -i.e. Left, is simplistic. Especially if this is to be understood simply by reading a text. The categories "Religion" and "Ideology", as mentioned above, have been created by grouping all the values in 4 major labels: non Believer, Christian, Muslim, other; and Left, Center, Right, Other. In order to improve the performances of both model one could think on editing the way the values are grouped, maybe creating more labels. One example could be to remove "Other" label and to create more small groups, as Buddhist, Hindu, Jewish for religion, or to keep all the ideologies of the users without grouping them. In fact, if no grouping is done for the Ideology, the class distribution will look like in figure 10. There are still some values that are closer to 0%, so the imbalanced class problem is not solved. Another way to improve the performance of the model, could be to solve the imbalance problem when training the BERT model, in order to see if it affects the values of the F1 scores. Furthermore, one of the motivation of a small value of the performance metrics in BERT Model, mostly for such sensible labels as Ideology and Religion, could be caused by the truncation hyper parameter, that is set to "longest first" while tokenizing. This parameter determines how the truncation is performed, and in this case, it compares every token in the debates titles and in the users' side, comparing the length of the tokens in these variables, choosing to keep the longest one. Due to this process, considering that in side variable the values are only Con/Pro, they can be neglected in order to give space to longer words in the debates' title variable. In this case there is a loss of information on the stances of a users regarding a certain topic, increasing the confusion of the model when classifying.

*6.0.1 Will the scores in SVM be higher if the whole text per users is used?* Considering that a data set, keeping every debates' title where

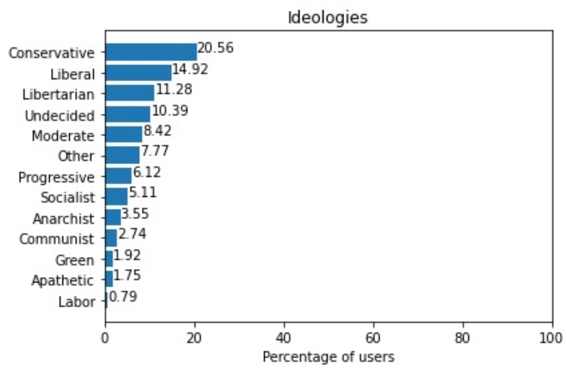


**Figure 10.** Class Distribution of ideologies

users participated in and their sides per each debate, contains only the meaningful information, it is fair to ask if, using the texts of the debates in which users joined, would lead to higher F1 Scores. This has been first checked with the BERT model while keeping max length set to 22, leading to no meaningful behaviour and no surprising changes. For this reasons and for sake of smoothness, it has been decided to omit the analysis. Moving forward, in order to investigate on the effects on the performance of adding also the arguments of users when training the SVM, the texts that each user wrote is taken into account, and comments have been tokenized and vectorized in order to give it as an input to the SVM MultiOutput Classifier trainer. A 10-fold cross validation has been applied, that generally results in a less biased or less optimistic estimate of the model skill than a simple train/test split. The important thing is that each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model 9 (k-1) times [6].

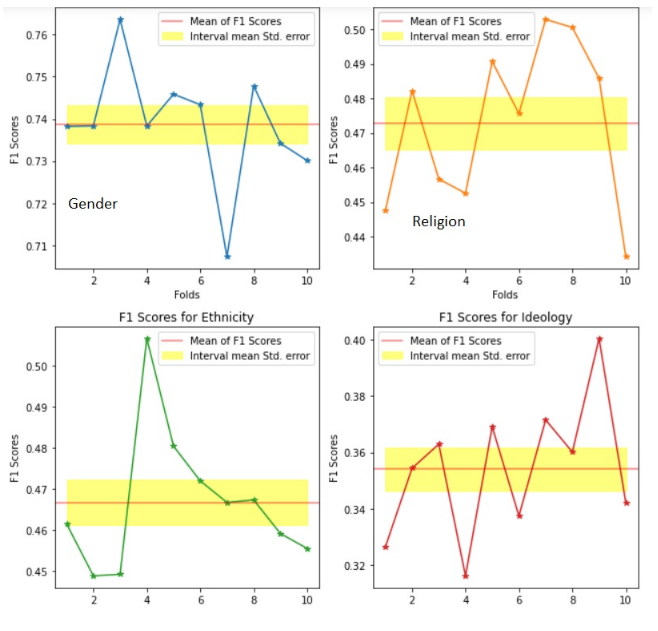


**Figure 11.** 10-Flods Cross Validation per each group

As shown in figure 11, the F1 means scores over all the folds are highly better for classifying users in their religion and gender, while the prediction for ethnicity and Ideology are just slightly better. Overall, for the performance of the SVM, in order to have a better classification, it is better to use also the texts that user wrote. This is logical, also because it's more likely that people argument more when debating on some topics, so more information may come out. It is also astonishing how the performance in this case are also better than the ones from the BERT model. This has happened maybe because in the BERT model, the imbalance in not taken into account while training, while in the SVM is included in the hyper-parameters.

Having seen that with more textual information the SVM performs better, one question could pop in mind: is the length of the text crucial for classifying users into groups? To investigate this hypothesis, the data set has been divided into two main data sets with respect to the length of the text that each user wrote. In order to split it, the median length has been computed $\mu_M = 8990$, so that one data set contains only the users who wrote text shorter than the median length text or equals to it, and the other one contains users who wrote texts longer than the median length text. After splitting both data sets in train (70%) and test (30%), an SVM trainer has been applied to them, leading to the prediction on the test with the F1 scores showed in figure 12.

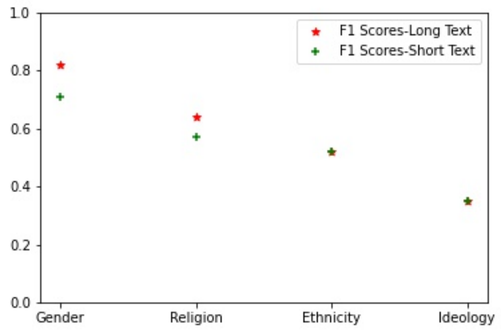


**Figure 12.** Comparing F1 Scores for users with text with $l \leq \mu_M$ and $l > \mu_M$.

The main takeaways are that for classifying users in gender and religion the length of the text is important, probably with longer text there are more clues on the gender of a person, and the user could explain more about his/her beliefs. Incredibly, the scores for the ethnicity class have increased with respect to the first SVM using the whole data set. One explanation could be that, with few and clear text examples of the ethnicity, it is easier for the model to not get confused about the ethnicity of some users. The ideology stays always about the same and it increases neither with longer text from users nor with shorter texts.

## 6.1 Implementing Multi Label BERT Classifier

One of the main aspects of the BERT model implemented before, is that it is trained on only one label per time, it doesn't predict the different labels contemporaneously. This can be a disadvantage because there can be some inner correlations among the labels themselves, that can help the model to predict the classes correctly. In order to investigate more on this idea, a Multi Label BERT model has been implemented, in order to predict, using as a covariate the debates' titles and the users' sides, all the labels at once. First of all, for every class, a dummy variable is created, tin order to have binary values to feed to the model. The text is tokenized in the same way as explained in the previous sections. The distributions of the tokens per each class is shown in the figure 13.

The tokenized text is then wrapped in a PyTorch Dataset containing the tokenized text, the input_ids and the attention_mask, along with converting the labels into a FloatTensor, that contains all the information on the classes. Secondly, using the library PyTorch Lightning [7], the custom data set is wrapped into LightningDataModule -i.e. a collection

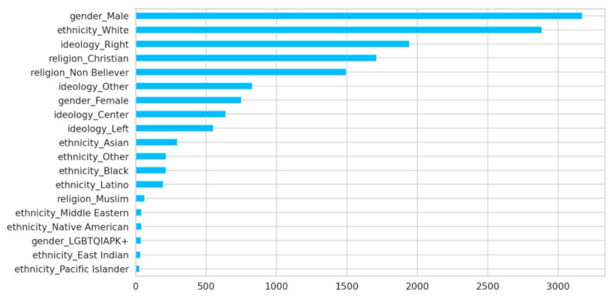


**Figure 13.** Number of tokens per each class.

of a train_dataloader(), val_dataloader(), test_dataloader() along with the matching transforms and data processing/downloads steps required; it encapsulates all data loading logic and returns the necessary data loaders, keeping the original percentage of train, evaluation and test data sets (80%, 10%, 10%). In order to train the model using the pre-trained BertModel and a linear layer to convert the BERT representation to a classification task, a LightningModule in implemented. This module organizes the PyTorch code into 5 sections: Computations, Train loop, Validation loop, Test loop and Optimizers, that is AdamW in this case. The optimizers is a scheduler that changes the learning rate of the optimizer during training. This might lead to better performance of the model. To use the scheduler, the number of training and warm-up steps must be computed (batch size is set to 12 and the epochs are chosen to be 3):

$$\#\_Training\_Steps\_per\_Epoch = \frac{\#Training\_examples}{Batch\_size} \tag{3}$$

$$\#\_Total\_Training\_Steps = \frac{\#\_Training\_Steps\_per\_Epoch}{\#\_Epochs}$$

Multi-label classification boils down to doing binary classification for each label/tag, and for this, in the model implementation, to measure the error for each label, the Binary Cross Entropy is combined with a sigmoid function. Finally, the model is ready to be trained and a standard pipeline with a checkpointing that saves the best model and with an early stopping triggers when the loss hasn't improved for the last 2 epochs, has been implemented. Once the training is completed, some evaluations on how the model performed are carried out, mostly by taking all predictions and labels from the validation set, in order to check the AUC of the ROC for each label. AUC stands for "Area under the ROC Curve", that is the entire two-dimensional area underneath the ROC curve from (0,0) to (1,1). It is an aggregate measure of performance across all possible classification thresholds and it can be interpreted as the probability that the model ranks a random positive example more highly than a random negative example. ROC is a probability curve and AUC represents the degree or measure of separability [8] and it tells how much the model is capable of distinguishing between classes. The ROC curve is plotted with TPR (True Postive Rate - the measure of the model of correctly identifying True Positives) against the FPR (False Positive Rate - the percentage of false positives compared to all positive predictions) where TPR is on the y-axis and FPR is on the x-axis. The AUC values of each label are displayed in figure 14.

The more the AUC value is near to 1, the better is the performance of the model into classifying text in these labels; and so, if the value is lower than 0.5, the performance is not so good. In this case, the labels with the AUC values higher than 0.5, are: for the religion Non Believer and Christian; for the ethnicity Pacific Islander, Black, Asian; for the ideology Right, Other, Left; and for the gender Male and Female. This means that the model can, overall, recognize very good some of the

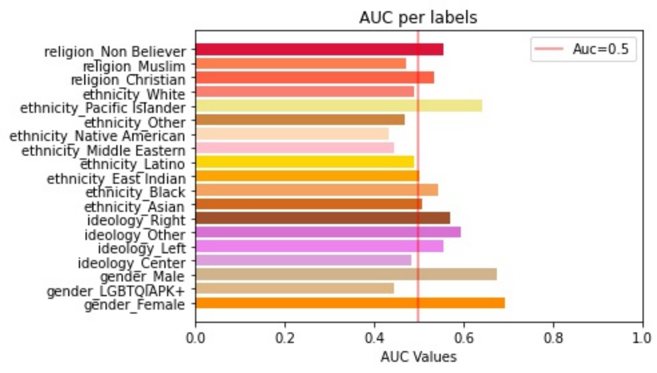


**Figure 14.** AUC values per each class.

labels in the previous macro-classes, as for example, ethnicity, but not all of them good. Most important thing is that the model can recognize 3 out of 4 labels of the ideology, increasing the performance compared to the single BERT model per single macro-category. One hypothesis is that these labels, taken each one per each one, are correlated with other labels, as for example, black people are more likely to be in leftist parties, because those parties have more anti-racial rules and ideas. Also, the number of tokens in the classes that scored lower than 0.5 is very small compared to the other (figure 13), and this is a big obstacle to the model while it's training, because it does not have enough examples to train with.

Furthermore, predictions using the test data set have been done, in order to compare them with the real values of the data set and the results are shown in the figure 15. The overall accuracy of the data is $acc_{BERT} = 0.83$, that is not bad, but considering the fact that data are highly imbalanced is not reliable.

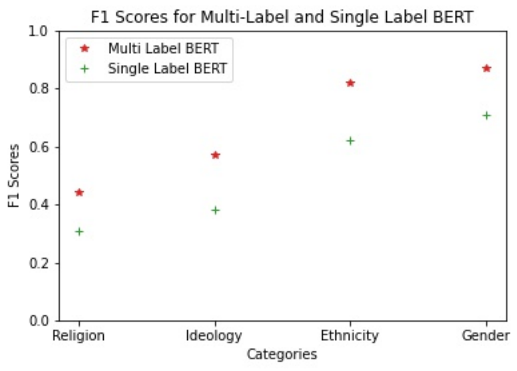


**Figure 15.** F1 Scores to compare multi label BERT and single label BERT.

The overall F1 scores per each category is plotted in figure 15. In order to have a better comparison between the model, the mean of F1 scores of the labels in each category is computed. As the plot shows, as expected when implementing the Multi Label BERT model, the overall performance is improved, even if for the Religion, F1 scores still lower than 0.5. The reason is still probably the one mentioned above when analyzing the performance of the single label BERT model, and it could be verified by feeding to the model all the text that users wrote without any truncation (in this project it is fed to the model with max length equals to 22), but for this, more powerful GPU and RAM are needed.

## 7 CONCLUSION

As social media becomes increasingly central to people's everyday lives, it is important to acknowledge how much can be understood by using NLP tools and libraries, and how well they can perform on those data. By handling the crawled data from the debate.org website, some decisions on which data to keep have been carried out. It has been decided to collect the titles and the text of the debates in which users joined and their sides per each debate, and to classify them into groups as religion beliefs, political ideas, genders and ethnicity. The data set is highly imbalanced due to the presence of more, for example, Christian Male. The analysis is carried out on two parallel ways: training the BERT model on a 80% train imbalance set, creating a baseline model, and training the SVM on the same data set, but taking into account the imbalance problem. In order to fine/tune the BERT model, some investigations on the padding and truncation hyper-parameters on the titles column are performed, mostly analyzing which is the max length that the tokenizer uses and how it affects the performances on the model. The results are that with a max length higher than 22 the performances are constant in their scores, so it has been decided, in order to reduce computational costs, to use 22 as a max length. The performances are analyzed by predicting the group of the users training on the titles and sides, resulting, in general, in high scores when predicting gender and ethnicity, but poorer scores when predicting religion beliefs and political ideas. To try to improve the performance of both models, the texts that users wrote in each debate, are used to train the models, resulting in a substantial improvements in the SVM metrics, but in a no significant change for BERT model. The enhancements are visible in the classification of every group, except for the political ideas, that remains with low scores. Furthermore, in order improve the performance of the BERT mode, dummyfication is applied to labels, that are concatenated together in order to build a Multi Label BERT model, using Pytorch Lightning. As a result, all the metrics reporting the performance have increased, denoting the implementation of a better model. Considering that in the BERT model the imbalance of the data set has not been taken into account, and still his performance is pretty elevated, at the same level of a Support Vector Machine that takes care of that, if the unbalanced data is kept into account and solved, maybe over-sampling with SMOTE [9], the BERT model could be a powerful tool to achieve the goal of finding the stakeholders groups in argument data.

## REFERENCES

[1] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[2] Hugging face library. Retreived from `https://huggingface.co/`. Accessed: 2022-02-16.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[4] Shan Suthaharan. Support vector machine. In *Machine learning models and algorithms for big data classification*, pages 207–235. Springer, 2016.

[5] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[6] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[7] Pytorch lightning library. Retreived from `https://www.pytorchlightning.ai/`. Accessed: 2022-02-16.

[8] Sarang Narkhede. Understanding auc-roc curve. *Towards Data Science*, 26(1):220–227, 2018.

[9] Jeniffer David, Jiarong Cui, and Fatemeh Rahimi. Classification of imbalanced dataset using bert embeddings. 2020.