# Identification of Human Values behind Arguments
## NLP Standard Project

**Matteo Nestola, Simona Scala** and **Sara Vorabbi**
Master's Degree in Artificial Intelligence, University of Bologna
{ matteo.nestola, simona.scala6, sara.vorabbi }@studio.unibo.it

### Abstract

The aim of this study is to identify human values categories from a textual argument using the methodology proposed in (Kiesel et al., 2022). The task involves multi-label classification, where 20 values need to be predicted. To achieve this, three pre-trained models for sequence classification, namely *bert-base-uncased*, *roberta-based*, and *allenai/multicite-multilabel-sciber*, are fine-tuned using the Hugging Face library. The experiment is repeated twice, once with the input composed only by premises and the second time where conclusions, stances and premises are concatenated. From the results obtained by the f1-score computed on the test set, it can be noticed that despite the additional knowledge present in the concatenated input, the models obtain worse performance compared to the models that take only the premises as input.

## 1 Introduction

Detecting human values in text, such as to have freedom of thought or to be broad-minded (Kiesel et al., 2022), is a crucial task in Natural Language Processing. The importance of human values has been studied for decades in social sciences and formal argumentation (Kiesel et al., 2022). According to the social sciences, a value is a belief about desirable end states or modes of conduct that guides behavior and is ordered by importance relative to other values to form a system of value priorities.

Identifying values in arguments can be difficult due to their large number, implicit use, and vague definitions. However, recent advancements in natural language understanding, the creation of larger argumentation datasets, and the taxonomization of values by social scientists have made automatic identification possible.

Unlike (Kiesel et al., 2022), this paper presents a multi-level taxonomy of 20 human values and a dataset (Mirzakhmedova et al., 2023) of 9324 arguments from 6 different sources, covering religious texts, political discussions, free-text arguments, newspaper editorials, and online democracy platforms.

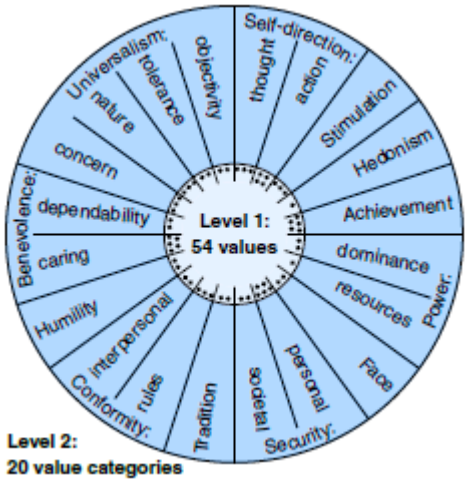


Figure 1: Capture of the labels

We based our experiments on the existing code presented in the GitHub repository of ACL'22 publication "Identifying the Human Values behind Arguments". In fact, we implemented a first model starting from the same *bert-base-uncased* checkpoint of the Hugging Face Library. Moreover, we added two other pre-trained models for sequence classification to compare the results obtained, namely *roberta-based* and *allenai/multicite-multilabel-sciber*. These models undergo a fine-tuning phase with specific hyper-parameters such as learning rate, number of training epochs, and batch size, to adapt them to the problem.

In the specific task at hand, an input can be assigned multiple labels simultaneously, which make it a multi-label classification problem. In particular, 20 different labels need to be predicted for a given input. The structure of the input itself is discussed in depth in Section 3. We conducted two separate experiments: in the first experiment, we used only

the premises of the input to make our predictions. Instead, in the second experiment, we set the input as the concatenation of conclusions, stances and premises in order to try to achieve better results.

The fine-tuned models are evaluated on F1-score, Precision, and Recall, averaged over all value categories. The models are ranked according to their averaged F1-score on the test dataset, particularly considering the presence or absence of concatenation between premises, stances, and conclusions.

Our results show that even though the dataset was bigger, extending the one used in (Kiesel et al., 2022), the performance obtained in our experiments are worse compared to the ones obtained in (Kiesel et al., 2022). The reason behind such results could be found in the dataset since the distribution of the classes appears to be very unbalanced, as explored in detail in Section 3, thus making the classification task very hard for all those classes for which the examples are missing. Moreover, even if the input concatenation should add knowledge, our results seems to prove that the performance worsens. It might be true that the input concatenation could add noise.

So we can conclude that this task of detecting human values in textual arguments poses unique challenges due to the informal and noisy nature of the text.

## 2 System description

The system consists of several essential steps, each of which is necessary to achieve the final goal:

### 2.1 Data loading

The necessary data files for our research are downloaded using the platform Zenodo, which hosts three sets of data files for training, validation, and testing. These files contain arguments and their associated labels and are stored in separate tab value (TSV) files. We use the `pd.read_csv()` method of the pandas library to read the set of arguments and labels as a dataframe. We specify the file paths for each set of data to ensure that they are loaded correctly. Using this approach, we are able to access the necessary data files and read them into a format that can be easily analyzed and used for model training and evaluation. This allows us to make our research process more efficient and to ensure that we have the necessary data to achieve our research objectives.

Since the arguments and the labels were in different file we merged the dataframes for each set using 'Argument ID' as the merging column and stores them in $df\_train$, $df\_val$, and $df\_test$ respectively. After this, we pre-processed the text with a pipeline described in the Data section. This pipeline is defined as a list of functions applied to each argument using the `text_prepare` function. These pre-processing steps ensure that the text is in a standardized format, making it easier to analyze and train models.

### 2.2 Pre-processing of the provided data

The input text undergoes pre-processing before tokenization. Special characters are removed. Multiple spaces are substituted with a single space. Stopwords are removed from the text, except for "against" and "favor," which are kept as they constitute the stance and will be used during the concatenation of arguments.
The following functions were taken directly from the second lab of the course:

- **lower(text: str) → str**: This function converts the given text to lowercase.
- **replace_special_characters(text: str) → str**: This function replaces special characters, such as parentheses, with a spacing character.
- **replace_br(text: str) → str**: This function replaces 'br' characters in the given text.
- **filter_out_uncommon_symbols(text: str) → str**: This function removes any special character that is not in the good symbols list, defined by a regular expression.
- **remove_stopwords(text: str) → str**: This function removes the stopwords from the given text, except for the words 'against' and 'favor', which are kept as they constitute the stance.
- **strip_text(text: str) → str**: This function removes any left or right spacing (including carriage return) from the text.
- **replace_double_spaces(text: str) → str**: This function replaces multiple spaces in the given text with a single space.

### 2.3 Input Tokenization

The tokenization step is computed by the `AutoTokenizer.from_pretrained()` method from the Hugging Face library only performs tokenization of the text input. It converts raw text input into a sequence of subwords or tokens.
By creating an instance of AutoTokenizer using the $from\_pretrained$ method, the tokenizer is initialized with the pre-trained weights from the specified model. Utilizing a pre-trained tokenizer can be advantageous for natural language processing tasks, as they have already been trained on large amounts of text data, and can accurately split input text into individual tokens while considering language nuances, such as punctuation, abbreviations, and other complexities. This eliminates the need to train a tokenizer from scratch, saving time and resources, and often results in improved performance for NLP models. Additionally, using the AutoTokenizer class allows for effortless switching between pre-trained models, without manual downloading and configuring of corresponding tokenizers, which is particularly useful when working on multiple projects requiring different models.

The method takes several parameters such as truncation, padding, and maximum sequence length, which affect how the text is tokenized and processed. The output will likely be a dictionary-like object containing various tokenization-related information, such as the token IDs, attention masks, and token type IDs.

### 2.4 Model Definition

The $AutoModelForSequenceClassification$ class provided by the Hugging Face library was used to select the model to be used. Specifically, this is a class of the Transformers library that automatically selects the appropriate model architecture based on the specified $model\_name$. The three pre-trained models, namely bert-base-uncased, roberta-base and multicite-multilabel-scibert were used to solve the task.
Here are the main differences between them:

- **bert-base-uncased**: This is a variant of the original BERT model that has been trained on a large corpus of uncased text. The model has 12 transformer layers, 768 hidden dimensions, and 110 million parameters. Because it is uncased, it treats all words as lowercase, regardless of their original capitalization. This can be useful for tasks where the case of the words is not important.

- **roberta-base**: This is a variant of the BERT model that was trained on a much larger corpus of text than the original BERT model. The model has 12 transformer layers, 768 hidden dimensions, and 125 million parameters. One major difference between roBERTa and BERT is the training data preprocessing: roBERTa uses dynamic masking and shuffling techniques, which are intended to improve the model's ability to handle out-of-domain data.

- **multicite-multilabel-scibert**: This is a variant of the SciBERT model that has been fine-tuned for multi-label text classification tasks. The model has 12 transformer layers, 768 hidden dimensions, and 110 million parameters. SciBERT was originally trained on scientific text, and the multicite-multilabel-scibert variant is fine-tuned on a multi-label dataset, meaning that each example can have multiple labels.

The models architecture are loaded using the `from_pretrained()` method from the `AutoModelForSequenceClassification` class in the transformers library.

## 3 Data

In the training and validation sets, each input is associated with a sequence of 20 binary digits, where 1 indicates that the input belongs to the corresponding label and 0 indicates that it does not. The Touché23-ValueEval Dataset is intended to investigate automated detection approaches for human values behind arguments.
The dataset comprises 9324 arguments from six different sources, including religious texts, political discussions, free-text arguments, newspaper editorials, and online democracy platforms. Each argument was annotated by three crowdworkers for 54 values.

The dataset consists of four tab-separated values files: arguments-training.tsv and labels-training.tsv for training, arguments-validation.tsv and labels-validation.tsv for validation, and arguments-validation-zhihu.tsv and labels-validation-zhihu.tsv

for testing.
The decision to use the zhihu files for testing was due to the unavailability of the labels-test.csv file.

Each `arguments-<split>.tsv` file has a header with four features: Argument ID, Conclusion, Stance, and Premise. Each row corresponds to one argument, where:

- `Argument ID` represents the unique identifier for each argument;
- `Premise` is the premise of the argument;
- `Conclusion` is the argument's conclusion;
- `Stance` is a flag that indicates whether the `Premise` is "in favor of" or "against" the `Conclusion`.

Each `labels-<split>.tsv` file has a header with multiple features, where the first one is `Argument ID`, used for matching with the data from the corresponding `arguments-<split>` file. Each column corresponds to one value category, with a value of 1 meaning that the argument employs the value category and 0 meaning that it does not.

The pre-processing pipeline is defined as a list of functions that are applied to the data (as accurately described above in section 2 "System description"). The `text_prepare` function takes a text string as input and applies the pre-processing pipeline to it, returning the pre-processed text string. The code applies the `text_prepare` function to the `Premise`, `Conclusion`, and `Stance` columns of the training, validation, and test dataframes, replacing each sentence with its pre-processed version.

The preprocessing pipeline was applied to all three datasets using the `apply()` function. The sentence length distribution was measured using a Box Plot, and a maximum length of 200 was determined through percentile mean calculation. Percentiles are commonly used in statistics and data analysis to measure the distribution of a dataset and identify extreme values or outliers. In this case, using percentiles to calculate the maximum sentence length helps to avoid bias towards extreme values and ensure that the model can handle the majority of sentences in the dataset without being overwhelmed by very long ones.

The training set exhibits class imbalance, with the label "Universalism: concern" having 2081 examples and the label "Hedonism" having only 172 examples.

## 4 Experimental setup and results

After selecting the three models outlined earlier, the subsequent step involves determining the hyperparameters to use for each model. These hyperparameters are values that control the learning process of the model and can be tuned to optimize its performance for a specific task. The process of selecting appropriate hyperparameters can greatly impact the accuracy and efficiency of the model, and requires a good understanding of the problem domain and the available data. In the AutoModelForSequenceClassification.from_pretrained class of the HuggingFace library, there are additional parameters that can be set in addition to hyperparameters. The "**num_labels**" parameter is used to specify the number of labels or classes in the classification task (it is set to 20, which is the number of labels in the dataset). On the other hand, the "**ignore_mismatched_sizes**" parameter is a boolean value that determines how the model handles input data that have different lengths. If set to True, the model will ignore data samples that have a different length than what was specified during training. If set to False, the model will raise an error when it encounters input data with a different length than what was specified during training. This parameter is useful in cases where the input data may have varying lengths, such as with text classification tasks where the length of the input text may vary (in this case it is set to True to ignore any mismatches in the size of input and output sequences).
These parameters are the same for all the chosen models.
The implemented models share several common hyperparameters. For instance, the **evaluation_strategy** and **save_strategies** were set to 'epoch', indicating that an evaluation is performed and the model is saved at the end of each epoch. The **learning_rate** was set to $2 \times 10^{-5}$, determining the speed at which the model updates its parameters based on the gradient of the loss function. The number of epochs was set to 20, but it depends on the Early Stopping settings, which can interrupt the training in advance in the event of overfitting.
Another interesting parameter is the weight de-

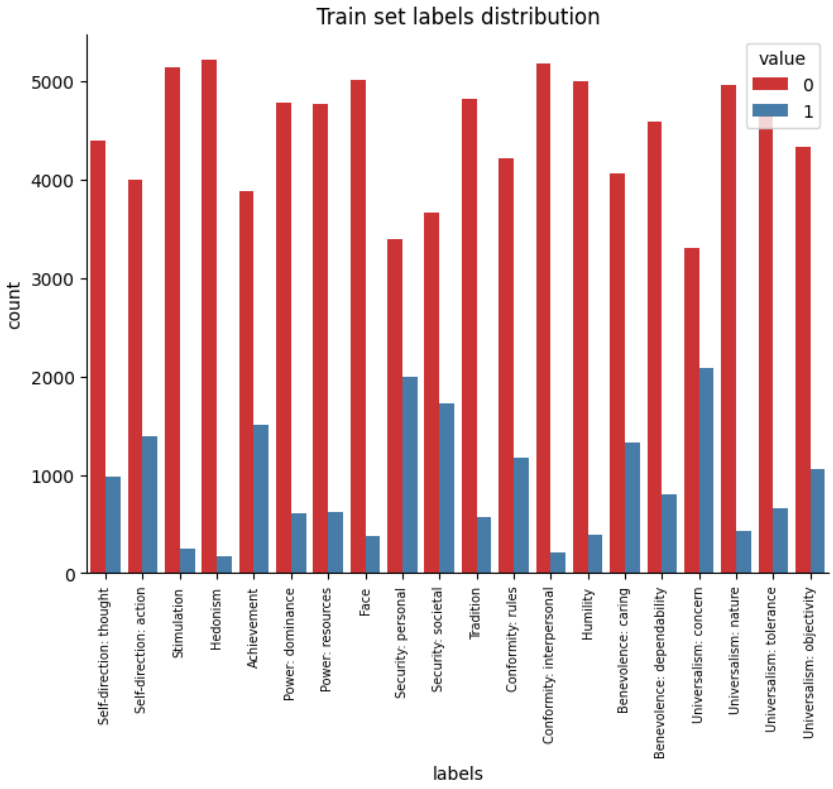


Figure 2: Distribution of the labels occurences in the training set

cay, a regularization technique that penalizes large weights in the model. In this case, the weight decay was set to 0.1, meaning the penalty is proportional to 0.1 times the square of the model's weights.
The Trainer object also allows for setting the batch size for both the training and validation sets: **per_device_train_batch_size** and **per_device_eval_batch_size**. After trying out different batch sizes, a batch size of 128 was found to provide the best performance for both parameters. Finally, **load_best_model_at_end** was set to True, indicating whether to load the best model at the end of training.

Another important step during the development process is the definition of the Custom Trainer class, which is inherited from the Trainer class of Hugging Face Transformers. The class allowed for customization of various training parameters such as the number of training epochs and batch size, to fine-tune the pre-trained models for the specific task at hand. Inside the Custom Trainer class, the Binary Cross Entropy With Logits Loss function was used as the loss function for the multi-label classification task. The Binary Cross Entropy With Logits Loss function assumes that the labels are encoded as binary indicators, where each sample can be associated with multiple classes, and expects the output of the neural network to be unnormalized logits, rather than probabilities. To normalize the probabilities, a sigmoid function was used. This loss function is commonly used for binary classification problems, and its use was crucial in achieving accurate multi-label classification of human values in textual arguments.

Each model was executed twice with different inputs: the first input concatenated the columns "Premise," "Stance," and "Conclusion," while the second input only considered the "Premise" column. The number of training epochs, which is dependent on the early stopping strategy is a critical hyper-parameter to consider.

The results in Table 1 and Table 2 show the performance of the three pre-trained models, namely bert-base-uncased, roberta-base, and multicite-multilabel-scibert, when applied to the task of classifying the stance of a given text with respect to a given topic. The evaluation metrics considered are precision, recall, and F1-score, which are common metrics for evaluating the performance of classification models.

The table shows that the best F1-score is achieved by the roberta-base model when the in-

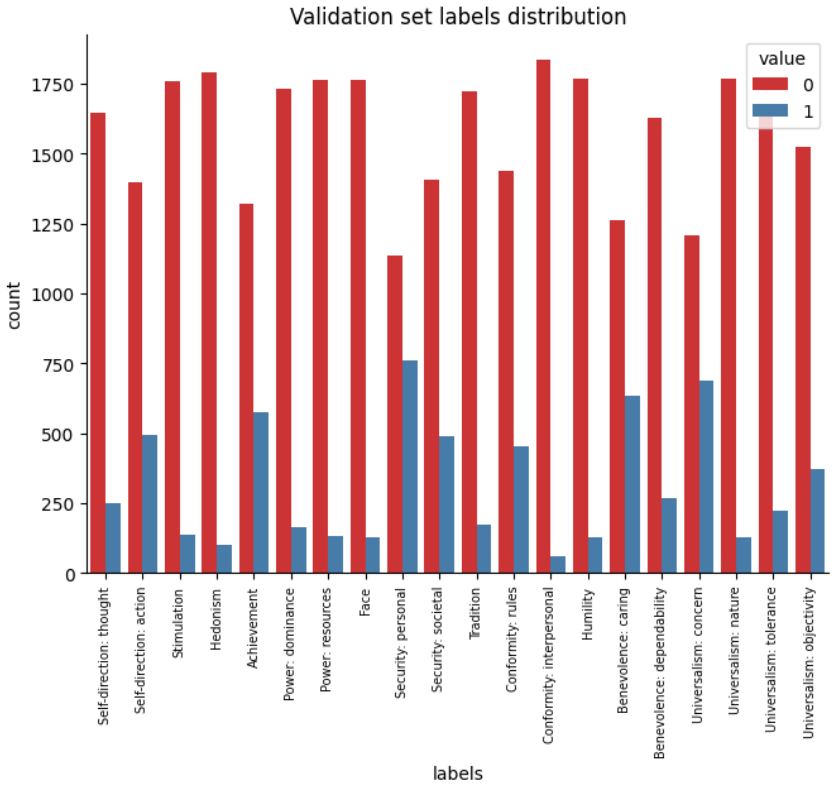


Figure 3: Distribution of the labels occurences in the validation set

put is not concatenated (i.e., only includes the "Premise" column) with a value of 0.3125, followed by the multicite-multilabel-scibert model with an F1-score of 0.2737. The bert-base-uncased model performs the worst, with an F1-score of 0.2676 when the input is concatenated and an F1-score of 0.2435 when the input is not concatenated. It's the only model to achieve better results when the input is concatenated.

It is worth noting that the best threshold value varies across models and inputs, which indicates that the performance of the models is sensitive to the threshold parameter. A higher threshold value leads to higher precision and lower recall, while a lower threshold value leads to higher recall and lower precision.

In summary, the roberta-base model achieved the best performance on this task, followed by the multicite-multilabel-scibert, while the bert-base-uncased model model performed the worst.

## 5 Discussion

The results shown in Table 1 indicate that the performance of the model during evaluation are not extremely satisfactory. Looking at the graphs of the training and validation datasets respectively shown in Fig. 2 and Fig. 3, it can be seen that the distribution of the data is quite similar. This could explain why the performances during evaluation we obtained better performances compared to the one obtained during prediction, as shown in Table 2. Indeed the F1-score is significantly lower in the case of prediction. Observing the graph of the test dataset in Fig. 4 it is possible to note that the data distribution differs from that of the train and validation datasets. Moreover the number of text arguments 9324 (Mirzakhmedova et al., 2023) may not be sufficient for a robust model training.

Analyzing in detail model by model, it can be seen that overall roBERTa seems to perform better than the other models. roBERTa is built on BERT, but with substantial differences, so the fact that it performs better may be due to these differences with BERT. Specifically roBERTa uses a different pre-train scheme, the training is performed with larger batches, and it uses BPE with bytes as a subunit and not characters (because of unicode characters) (Liu et al., 2019).

Allenai/multicite-multilabel-scibert, instead, was chosen because it is a model that does multi label classification. But a major setback is that it's trained on scientific papers, reason why it performs

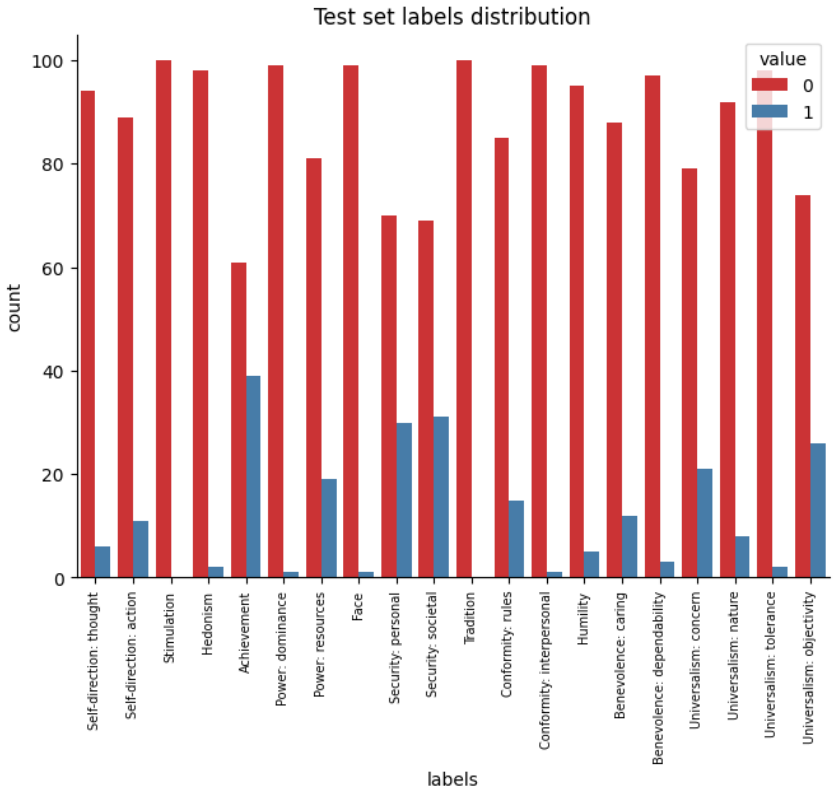


Figure 4: Distribution of the labels occurences in the test set

| Model | Precision | Recall | F1-score | Best Threshold | Concatenation |
|---|---|---|---|---|---|
| bert-base-uncased | 0.3059 | 0.5776 | 0.3790 | 0.15 | False |
| bert-base-uncased | 0.3271 | 0.5481 | 0.3681 | 0.15 | True |
| roberta-base | 0.3462 | 0.5892 | 0.4203 | 0.15 | False |
| roberta-base | 0.3327 | 0.6189 | 0.4246 | 0.15 | True |
| multicite-multilabel-scibert | 0.3176 | 0.5867 | 0.3948 | 0.15 | False |
| multicite-multilabel-scibert | 0.3060 | 0.5861 | 0.3855 | 0.15 | True |

Table 1: Results of the Evaluation

| Model | F1-score | Best Threshold | Concatenation |
|---|---|---|---|
| bert-base-uncased | 0.2657 | 0.15 | False |
| bert-base-uncased | 0.2732 | 0.15 | True |
| roberta-base | 0.2761 | 0.15 | False |
| roberta-base | 0.2752 | 0.25 | True |
| multicite-multilabel-scibert | 0.2704 | 0.20 | False |
| multicite-multilabel-scibert | 0.2672 | 0.30 | True |

Table 2: Results of the Prediction

worse than roBERTa.

In the majority of the cases, contrary to our expectations, we observed that models trained solely on the premises input performed better than those trained on concatenated input consisting of premise, stances and conclusions.

We speculate that this unexpected result could be attributed to the potential confusion that arises in the model when processing larger inputs. The complexity of concatenated input could overwhelm the model, leading to a loss of accuracy and reduced performance.

## 6 Conclusion

As previously discussed, we can conclude that the task of identification of Human values behind arguments can be very challenging. The dataset being small in size means that the model is not exposed to a diverse set of examples during training, leading to a lack of generalization ability. In addition to that, the unbalanced nature of the data means that some labels are much more prevalent than others in the dataset, and the models trained on such data tend to be biased towards the frequent labels.

In light of this, we cannot hide a certain surprise towards the obtained results, as we had assumed that the concatenated input might help the models better predict the labels.

## 7 Links to external resources

Webis - acl22 identifying the human values behind arguments https://github.com/webis-de/acl22-identifying-the-human-values-behind-arguments

Zenodo Touché23-ValueEval dowload https://zenodo.org/record/7550385

Human Value Detection 2023 https://touche.webis.de/semeval23/touche23-web/index.html

Tutorial 2 - Paolo Torroni, Andrea Galassi, Federico Ruggeri https://colab.research.google.com/drive/1j9XMnjEctcoQVEtbkRS0Zbm6FXOvyjtl#scrollTo=rduSZj7b5eiP

AutoTokenizer - Hugging Face https://huggingface.co/docs/transformers/v4.28.1/en/model_doc/auto#transformers.AutoTokenizer.from_pretrained

## References


Johannes Kiesel, Milad Alshomary, Nicolas Handke, Xiaoni Cai, Henning Wachsmuth, and Benno Stein. 2022. Identifying the human values behind arguments. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4459–4471, Dublin, Ireland. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Nailia Mirzakhmedova, Johannes Kiesel, Milad Alshomary, Maximilian Heinrich, Nicolas Handke, Xiaoni Cai, Barriere Valentin, Doratossadat Dastgheib, Omid Ghahroodi, Mohammad Ali Sadraei, Ehsaneddin Asgari, Lea Kawaletz, Henning Wachsmuth, and Benno Stein. 2023. The touché23-valueeval dataset for identifying human values behind arguments.