Identification of Human Values behind Arguments
NLP Course 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 the project is to identify human value 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, an automodel for text classification is implemented using the Hugging Face Library and pretrained language model checkpoints. Three different checkpoints are evaluated, namely bert-base-uncased, roberta-based, and allenai/multicite-multilabel-sciber, and their effectiveness in detecting human values in textual arguments is compared. The evaluation is based on macro f1-score, precision, and recall calculated on the validation set and macro f1-score computed on the test set.

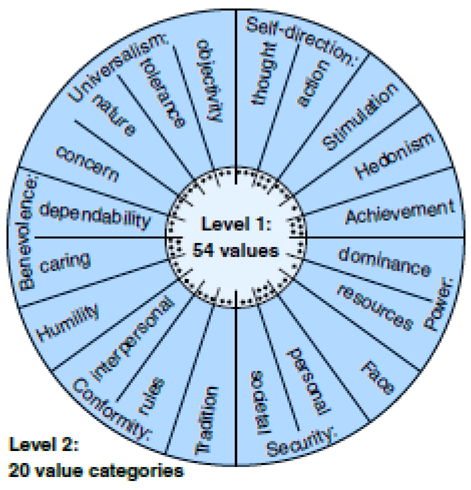

Figure 1: Capture of the labels

1 Introduction

Detecting human values in text is a crucial task in natural language processing with potential applications in multiple fields, including social sciences, psychology, and marketing. This task focuses on detecting human values in textual arguments, which poses unique challenges due to the informal and noisy nature of the text.

Each textual argument can be associated with one or more labels, making the task a multi-label classification problem, with 20 labels in particular for this task. The inputs are composed of Conclusion, Stance, and Premise, which can be concatenated to achieve better results.

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. To solve the multi-label classification task, three pre-trained models available in the Hugging Face library are used: BERT-Base Uncased, RoBERTa-Base, and 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 specific task.

The fine-tuned models are then evaluated on F1-score, Precision, and Recall, averaged over all value categories and for each category individually. The models are ranked according to their averaged F1-score on the test dataset, particularly considering the presence or absence of concatenation between Premise, Stance, and Conclusion.

In summary, the project focuses on multi-label classification to detect human values in textual arguments using pre-trained models, which are fine-tuned with specific hyper-parameters. The models' performances are evaluated based on F1-score, Precision, and Recall, and compared on the test dataset.

2 Background

The task at hand presents unique challenges due to the informal and noisy nature of the text. During the development of the task, challenges were faced, such as the absence of labeled data in the test dataset, which required the adoption of an additional validation dataset provided along with the other downloaded data. To improve the model's performance, three columns of the dataset, namely conclusion, stance, and premise, were concatenated instead of using only the column premise as input.

A Custom Trainer class was defined during the development process, which 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. Overall, the project focused on the use of pre-trained models for multi-label classification of human values in textual arguments and various challenges were overcome to achieve the goal.

3 System description

The system we have designed is composed of several essential steps, each of which is necessary for achieving the final objective:

1. data loading and initial analysis;
2. pre-processing of the provided data;
3. tokenization of text arguments:
   (a) choice of tokenizer;
   (b) selection of tokenizer parameters;
4. definition of the model(s);
5. training of the model(s);
6. evaluation of the model(s);
7. comparison of the models and selection of the best model.

3.1 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-separated value (TSV) files. We use the pandas library's $pd.read_csv()$ method to read in each 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.

3.2 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. 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.

3.3 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 $model\_name$.

3.4 Training of the model(s)

After defining the model, the following step consist of defining the *CustomTrainer*, a subclass of the Trainer class of the Transformers library that defines a custom training cycle for the multi-label classification task. The parameters of this model are defined via *TrainingArguments*, a class used to define all hyperparameters. The *CustomTrainer* function also takes as input a function that computes the metrics, callbacks such as *EarlyStopping*, the model and of course the training and evaluation dataset.

3.5 Evaluation of the model(s)

Through the *evaluate*() method of *CustomTrainer*, F1-score, precision and recall are calculated on the validation set.

3.6 Models comparaison

Considering the results obtained on the different models, that can be observed in Table 1 and in Table 2, the one with the best performance is chosen.

4 Data

The data used in this project was obtained from the Human Value Detection 2023 dataset. It consists of four .tsv 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.
Each argument in the argument files has a unique ID, a conclusion, a stance towards the conclusion, and a premise. Each label in the label files has a unique ID and 20 columns corresponding to value categories, with a 1 indicating that the argument belongs to that category and a 0 indicating otherwise.
The first preprocessing step was to merge the arguments with the labels to create a single dataset for training, validation, and testing. The training set consists of 5393 arguments, the validation set consists of 1896 arguments, and the test set consists of 100 arguments.
Other pre-processing steps are :

- Lowercasing all text.
- Replacing special characters such as parentheses, braces, brackets, and commas with spaces.
- Removing the string 'br' from the text.
- Filtering out uncommon symbols using a regular expression pattern that matches any non-alphanumeric characters except spaces, the '#' character, and the '+' character.
- Removing stopwords from the text using the NLTK library's stopword list for English, with the exception of the words 'favor' and 'against'.
- Stripping leading and trailing spaces from the text.
- Replacing multiple spaces with a single space.

The pre-processing pipeline is defined as a list of functions that apply each of the above steps in sequence. The text_prepare function takes a text string as input and applies the pre-processing pipeline to it, returning the pre-processed text string.
Finally, 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.
The training set exhibits class imbalance, with the label "Universalism: concern" having 2081 examples and the label "Hedonism" having only 172 examples. A Catplot from the Seaborn Library was used to visualize the label distribution.
Using a Catplot from the Seaborn Library, it was possible to visualize the results (2).

5 Experimental setup and results

The three pre-trained models, namely bert-base-uncased, roberta-base and multicite-multilabel-scibert were utilized 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

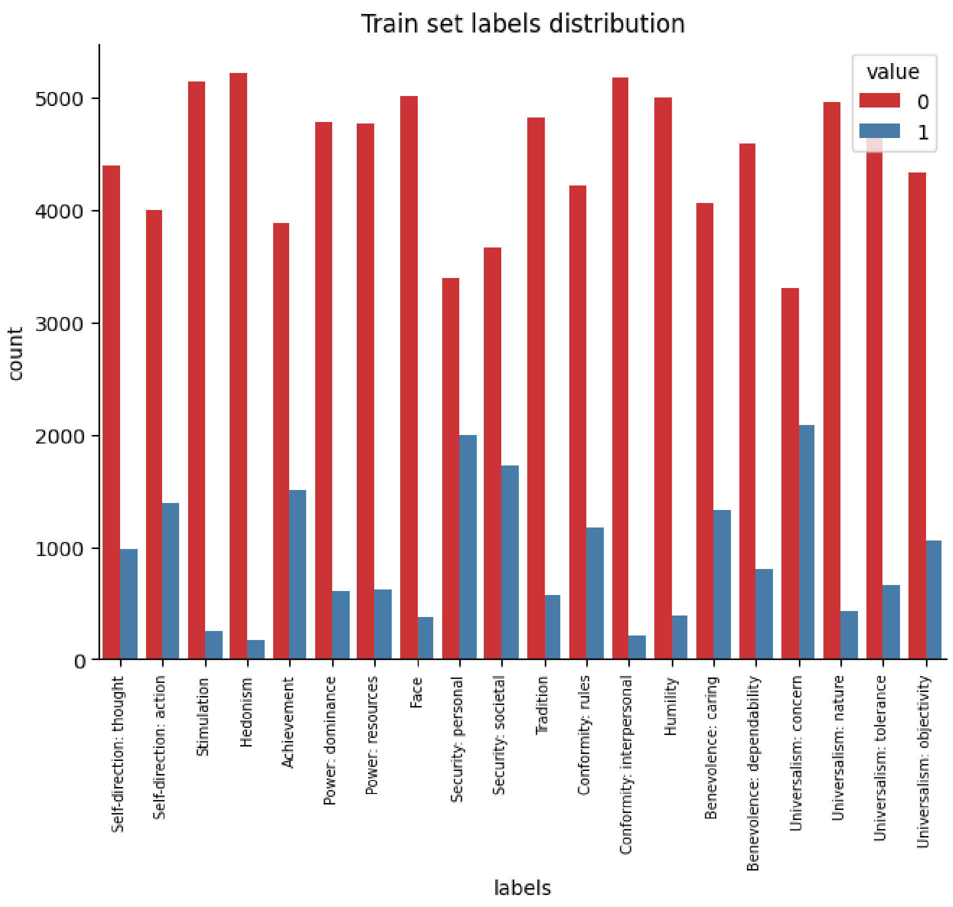

Figure 2: Distribution of the labels occurences

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.

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 hyperparameter to consider.

The implemented models share several common parameters. For instance, the evaluation 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 2×10^{-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 can be reduced based on the Early Stopping settings.

Another interesting parameter is the weight decay, 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.

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 input 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.

6 Discussion

From the three models used: bert-base-uncased, roberta-base and multicite-multilabel-scibert, the results obtained were as follows:

- bert-base-uncased (no concat) : F1= 0.2435 ,
- bert-base-uncased (concat) : F1= 0.2676,
- roberta-base (no concat) : F1= 0.3125 ,
- roberta-base (concat): F1= 0.2609 ,
- multicite-multilabel-scibert (no concat): F1= 0.2737 ,
- multicite-multilabel-scibert (concat): F1= 0.2576,

As we can see from the results, the F1 score of the models is not very high. This could be due to the fact that the size of the training dataset, since it does not contain many examples.
In the case of a multi-label classification problem with a smaller training set, RoBERTa should perform better than the other models thanks to its ability to learn more general language representations. In fact, with a smaller training set, it is essential to take advantage of the pre-trained representations to effectively learn from the limited data available.
RoBERTa is trained on a much larger corpus of text, reason why it could have learned a better and more robust representations that can be more easily adapted to the specific task.
Moreover, its dynamic masking and shuffling techniques may help the model to better generalize to new examples and perform well even on a small training set, like the one we are provided.

One of the key findings from our research is the unexpected result we obtained when comparing models with concatenated input to those with non-concatenated input.
One of the key findings from our research is the unexpected result we obtained when comparing models with concatenated input to those with non-concatenated input. However, contrary to our expectations, we observed that models trained solely on the "Premise" input performed better than those trained on concatenated input consisting of "Premise," "Stance," and "Conclusion."
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.
This finding highlights the importance of considering input size and complexity when designing and training machine learning models.

7 Conclusion

As previously discussed, the classification results for F1, Precision and Recall metrics are not satisfactory. This can be mainly attributed to two reasons

Model	Epochs	Precision	Recall	F1-score	Best Threshold	Concat
bert-base-uncased	12	0.3074	0.5610	0.3670	0.15	False
bert-base-uncased	12	0.3144	0.5614	0.3689	0.15	True
roberta-base	12	0.3708	0.5407	0.4253	0.20	False
roberta-base	12	0.3169	0.6027	0.4053	0.15	True
multicite-multilabel-scibert	11	0.3140	0.6040	0.3972	0.15	False
multicite-multilabel-scibert	11	0.2941	0.5640	0.3677	0.15	True

Table 1: Results of the Evaluation

Model	F1-score	Best Threshold	Concat
bert-base-uncased	0.2435	0.10	False
bert-base-uncased	0.2676	0.15	True
roberta-base	0.3125	0.30	False
roberta-base	0.2609	0.25	True
multicite-multilabel-scibert	0.2737	0.15	False
multicite-multilabel-scibert	0.2576	0.35	True

Table 2: Results of the Prediction

- the small size of the dataset and the unbalanced nature of the data with respect to the labels.

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. On the other hand, 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.

This issue leads to a situation where the trained models perform well on the labels with high occurrence in the training dataset, but their performance is very poor on labels with low occurrence.

As a result, the overall performance of the model is heavily influenced by the occurrence of the labels in the dataset.

This is especially true for multi-label classification problems where each example can have multiple labels associated with it.

In summary, the low performance of the models on this dataset can be attributed to the small size of the dataset and the highly unbalanced nature of the labels.

by analyzing inputs that correspond to frequently occurring labels, the model exhibits an ability to accurately identify the labels.

However, when the model is presented with a text argument that includes a label with a low frequency of occurrence, it is unable to accurately associate the input with the correct label.

8 Links to external resources

Zenodo Touché23-ValueEval dowload `https://zenodo.org/record/7550385`
Human Value Detection 2023 `https://touche.webis.de/semeval23/touche23-web/index.html`

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.