HUMAN VALUE IDENTIFICATION

MIGUEL CORREA

February 2024

AGENDA

INTRODUCTION

DATA

MODEL

METHODOLOGY

RESULTS

CONCLUSIONS

INTRODUCTION

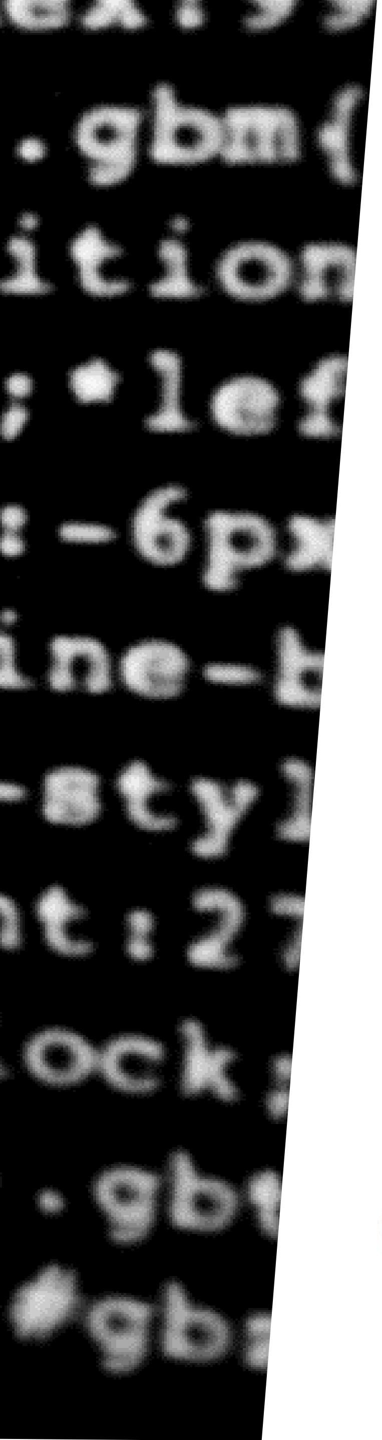
HUMAN VALUES IDENTIFICATION

What is the importance and Applications of Human Value Identification Algorithms?

Importance: Key to understanding cultural and social dynamics; crucial for the development of responsible and ethical AI technologies.

Applications:

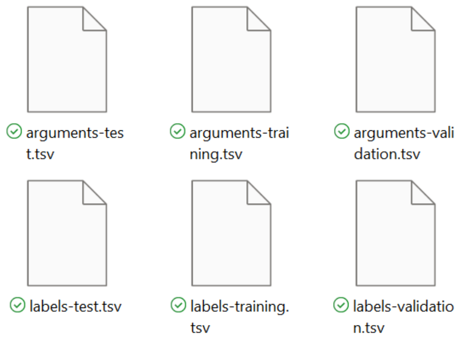
- *Ethical AI Development*
- Education
- Media Analysis
- Cultural and Social Intelligence
- Historical Analysis



DATA

SEMEVAL 2023 TASK 4

The data used for this challenge was provided by the challenge “SemEval 2023 Task 4. ValueEval: Identification of Human Values behind Arguments”




```
df_training.head()
```

	Argument ID	Conclusion	Stance	Premise	Self-direction: thought	Self-direction: action	Stimulation	Hedonism	Achievement	Power: dominance	...
0	A01002	We should ban human cloning	in favor of	we should ban human cloning as it will only ca...	0	0	0	0	0	0	...
1	A01005	We should ban fast food	in favor of	fast food should be banned because it is reall...	0	0	0	0	0	0	...
2	A01006	We should end the use of economic sanctions	against	sometimes economic sanctions are the only thin...	0	0	0	0	0	1	...

python + colab   pandas

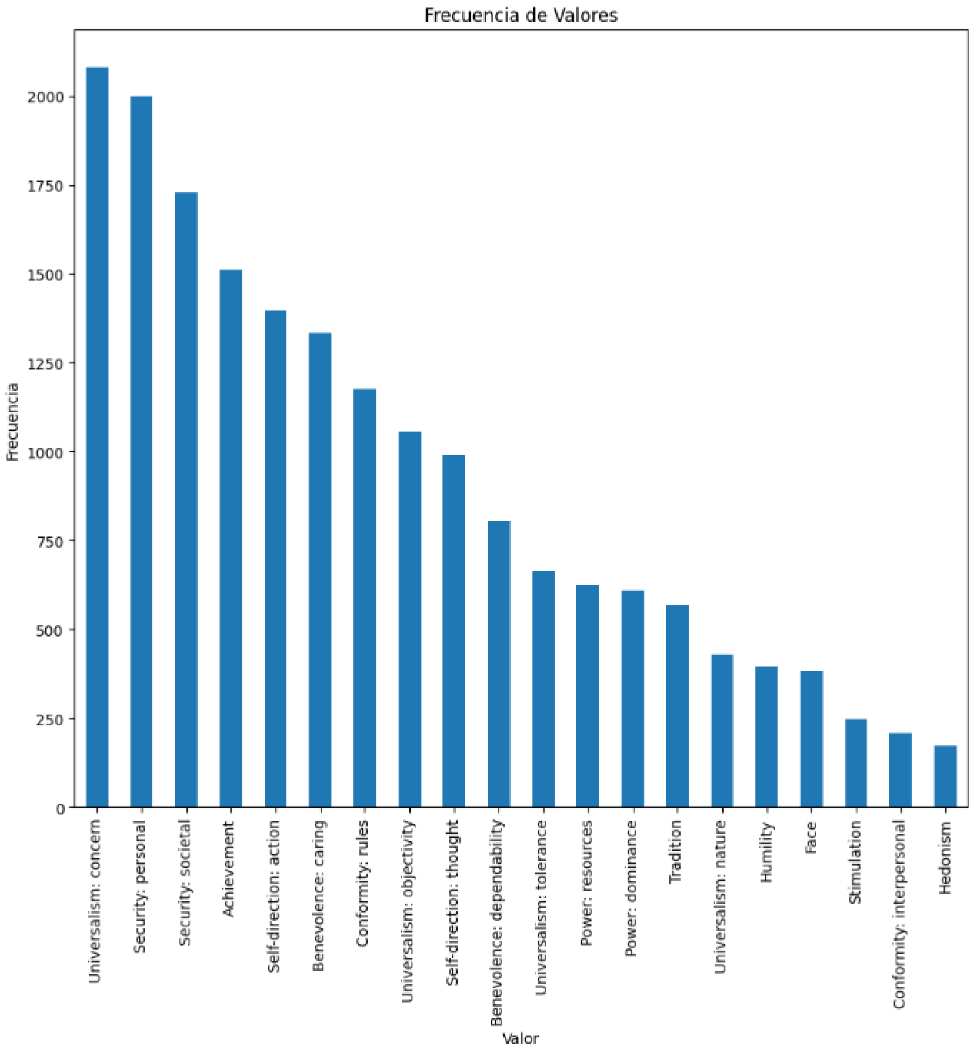
3 Columns with text
20 Labels

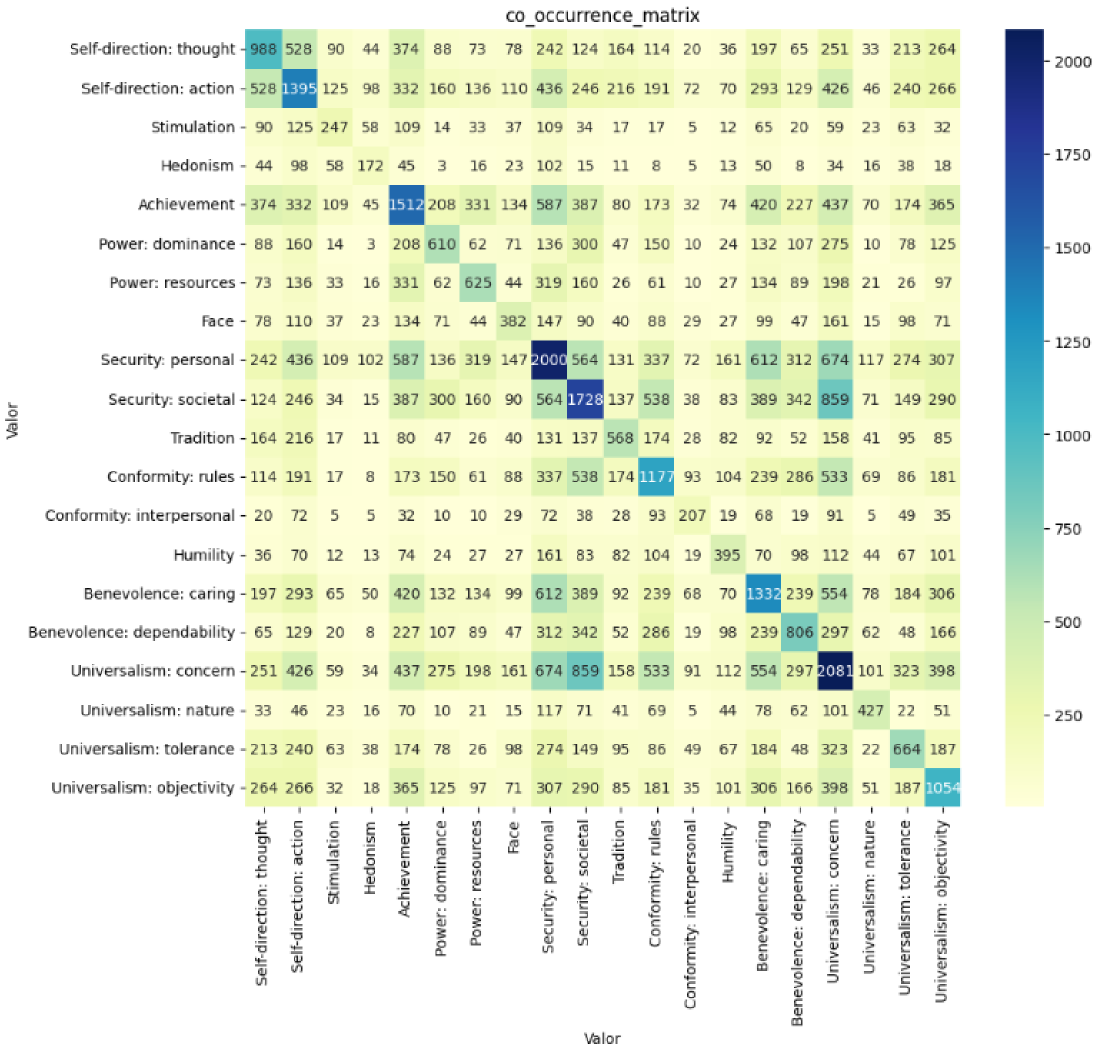
Taxonamy of the data

```
[8] import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    def plot_dataset(df):
      value_columns = df.columns[4:]

      frequencies = df[value_columns].sum().sort_values(ascending=False)

      # Analysis Co-ocurrences
      co_occurrence_matrix = df[value_columns].T.dot(df[value_columns])

      fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))

      # Visualization of frequencies
      frequencies.plot(kind='bar', ax=ax[0])
      ax[0].set_title('Frecuencia de Valores')
      ax[0].set_xlabel('Valor')
      ax[0].set_ylabel('Frecuencia')

      # Visualization of Co-ocurrences
      sns.heatmap(co_occurrence_matrix, annot=True, fmt="d", cmap="YlGnBu", ax=ax[1])
      ax[1].set_title('co_occurrence_matrix')
      ax[1].set_xlabel('Valor')
      ax[1].set_ylabel('Valor')

      # layout adjustment
      plt.tight_layout()
      plt.show()
```

TRAINING DATASET

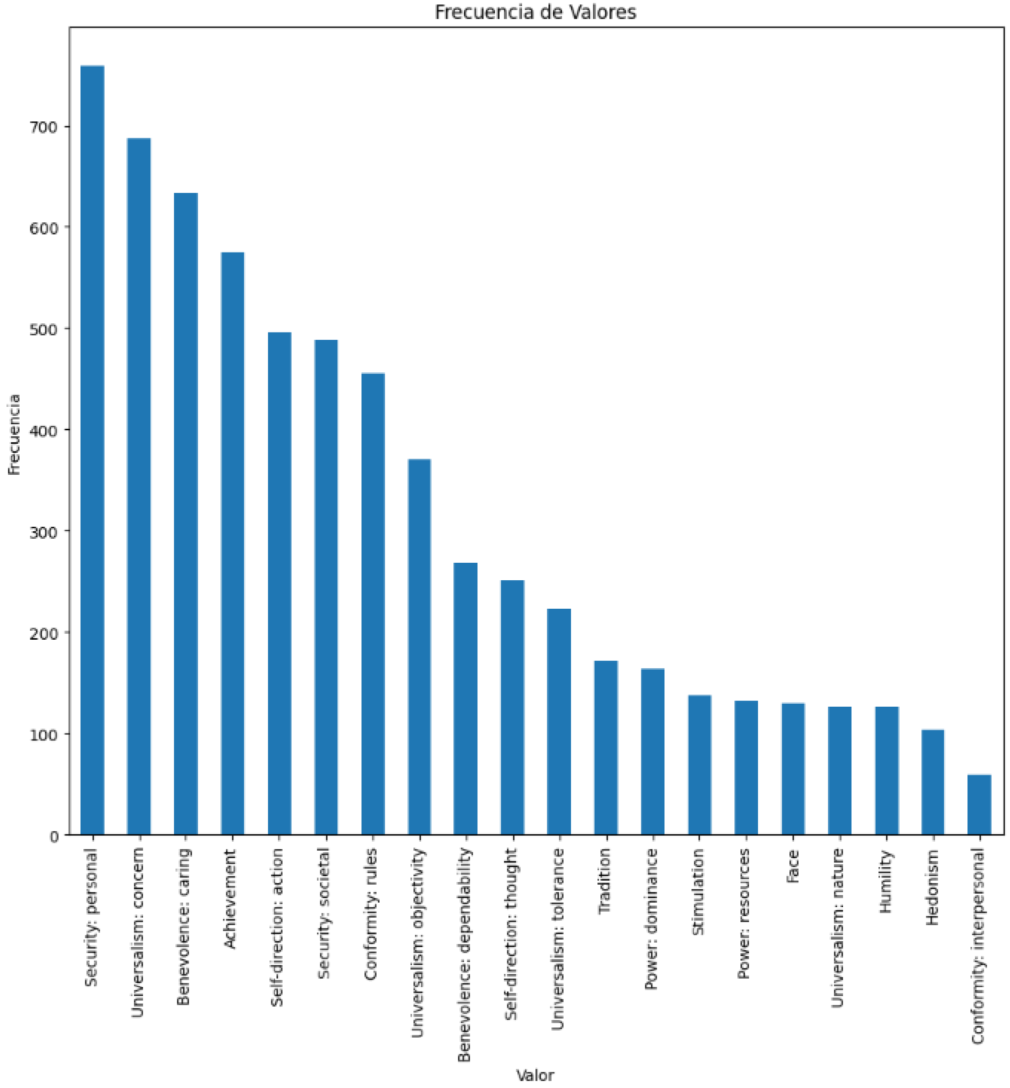
```
plot_dataset(df_training)
```

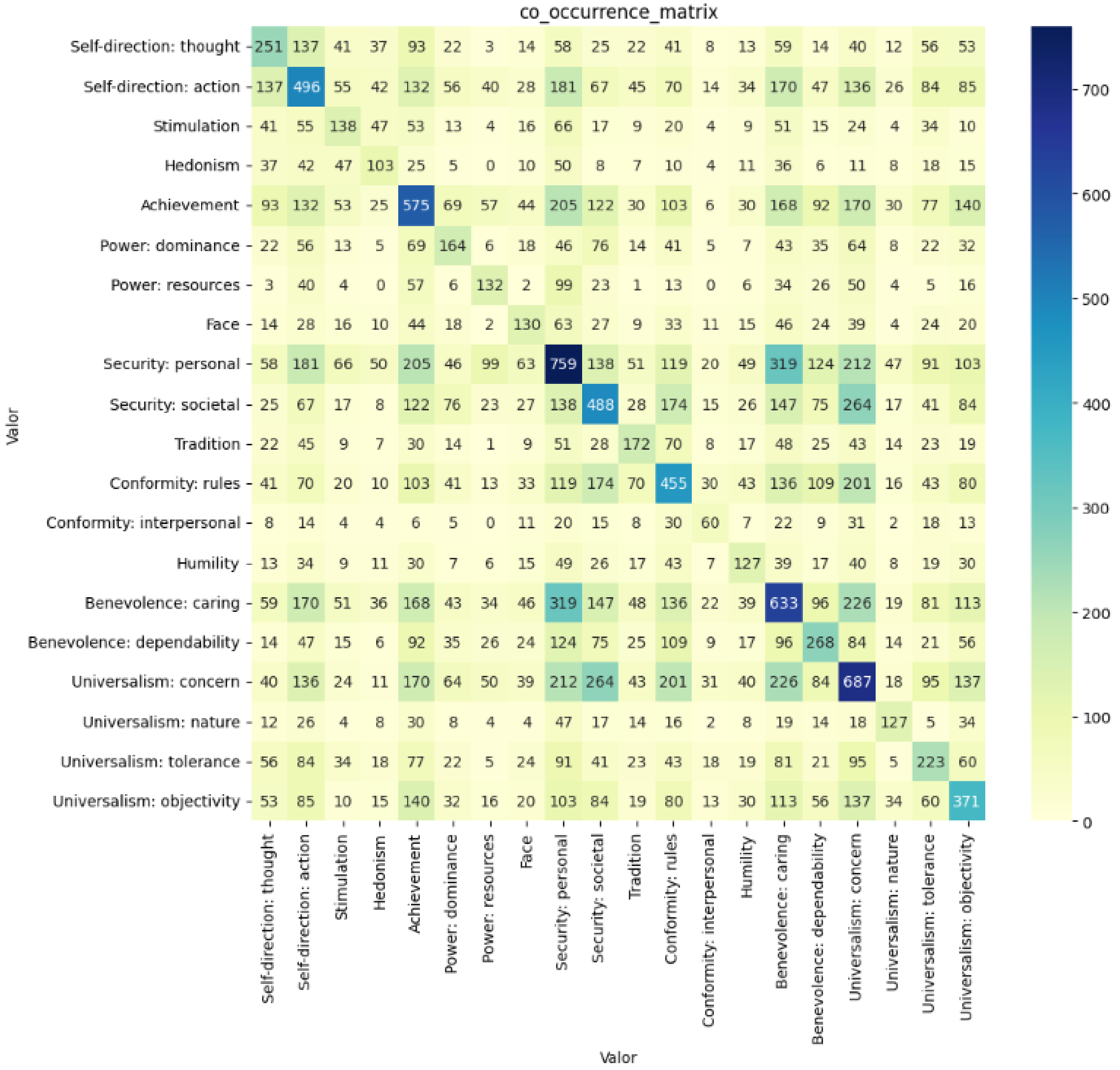




VALIDATION DATASET

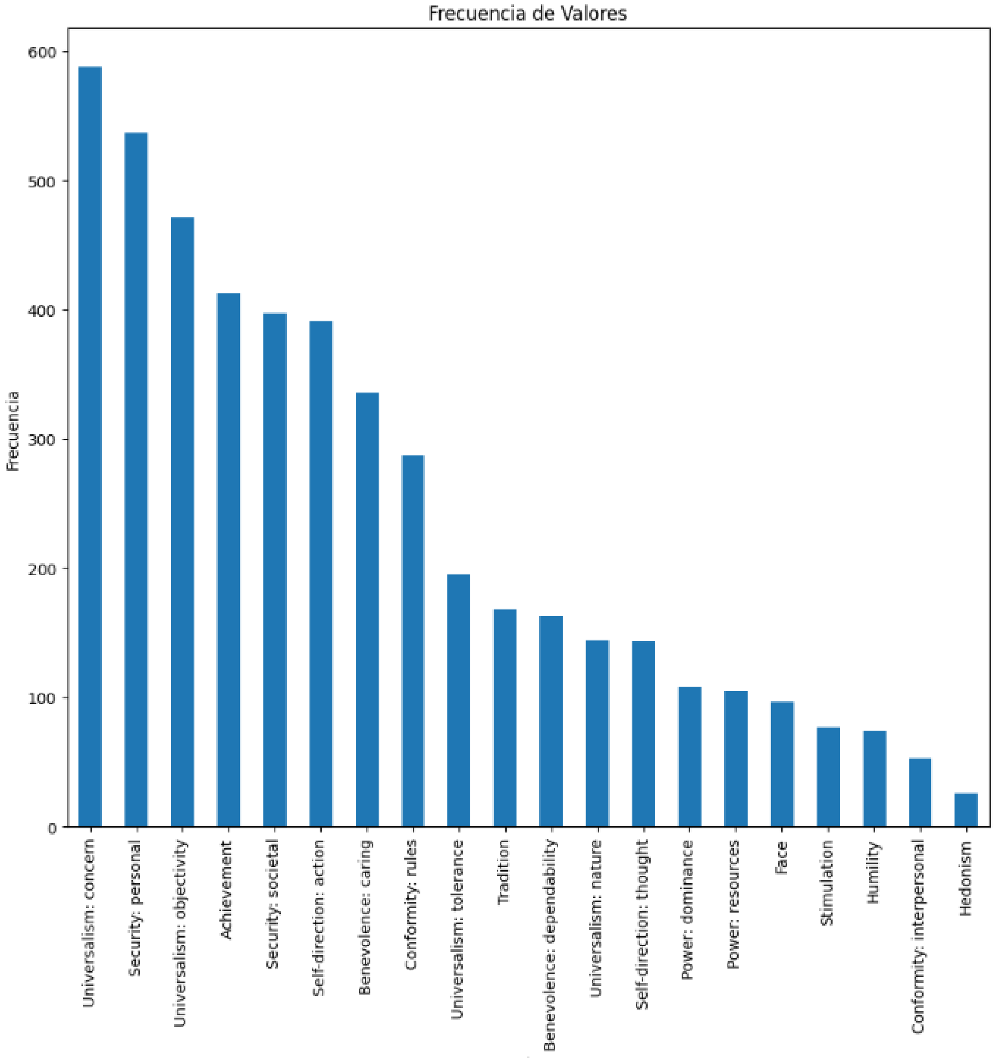
```
plot_dataset(df_validation)
```

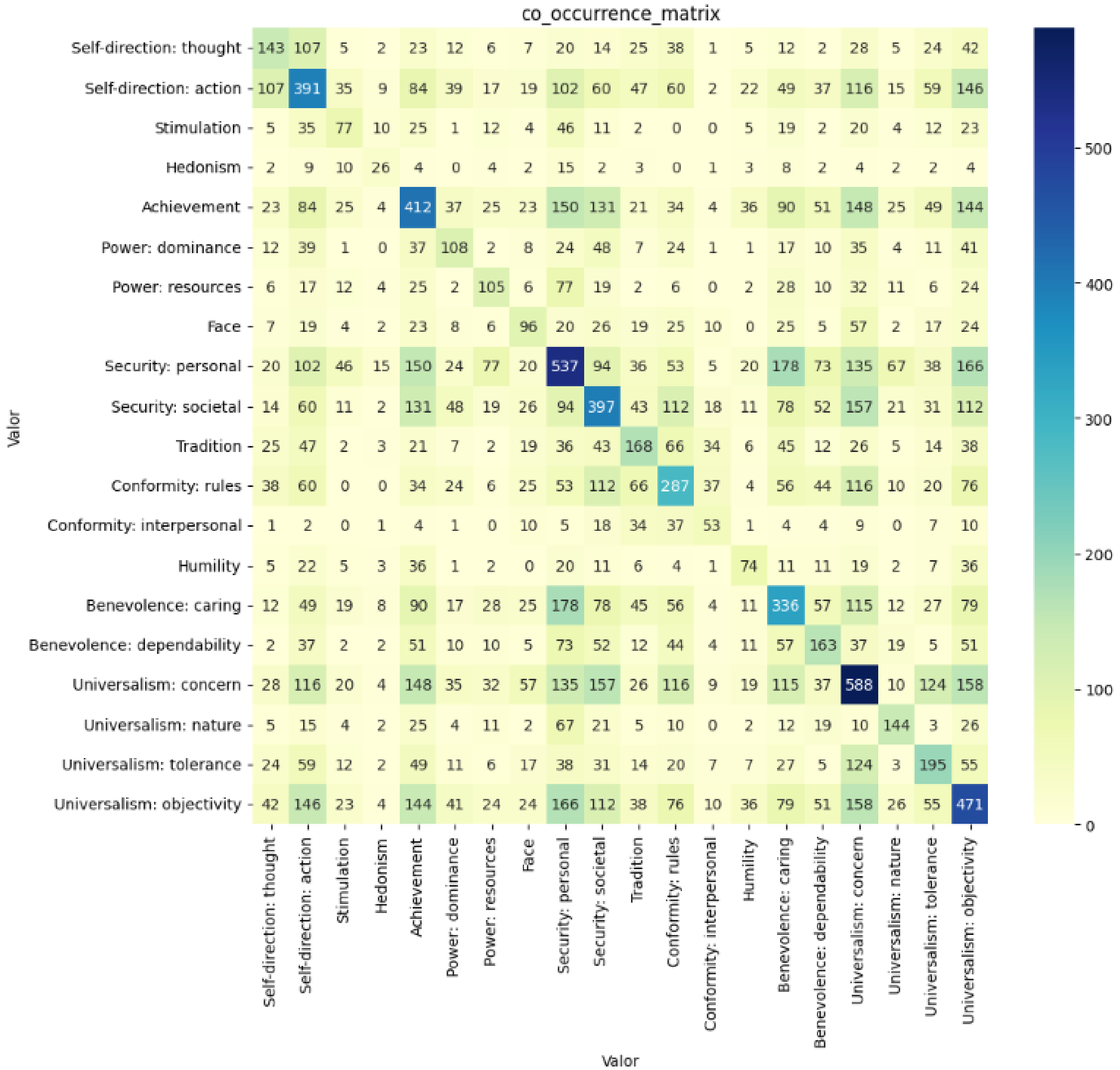


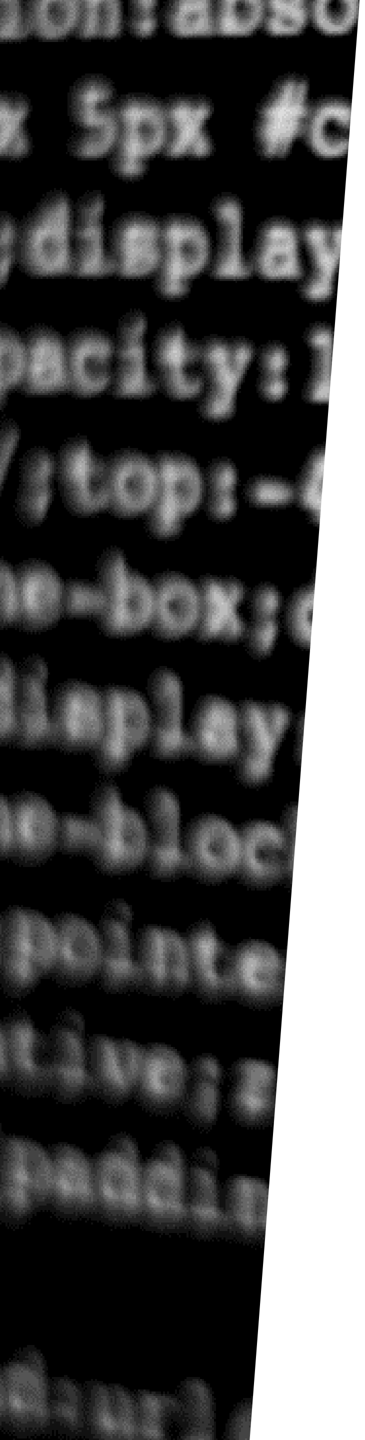


TESTING DATASET

```
[11] plot_dataset(df_test)
```





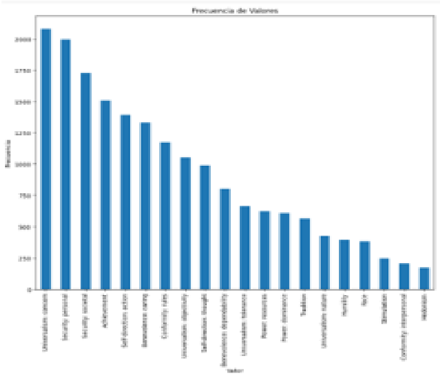


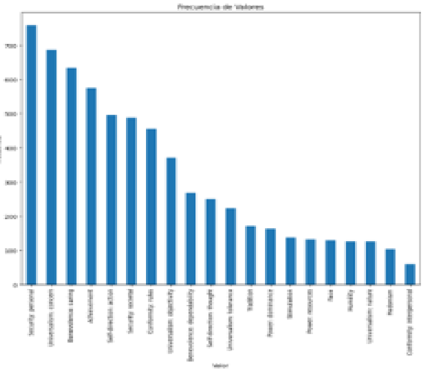
DATA

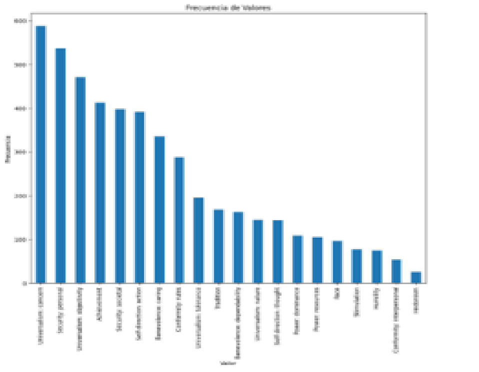
SEMEVAL 2023 TASK 4

Examining the distribution of samples for each label, we can see that there is not a uniform distribution across the number of samples among the labels.

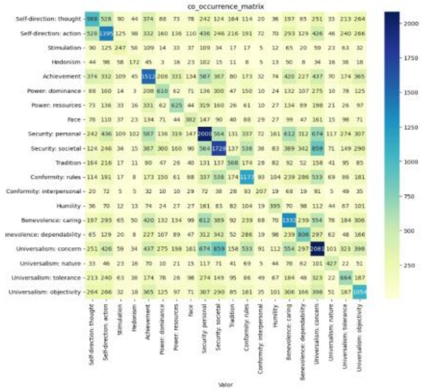
However, it is evident that across the different datasets, the distribution of labels remains relatively consistent according to the histograms of the labels, despite the varying number of samples in each dataset.

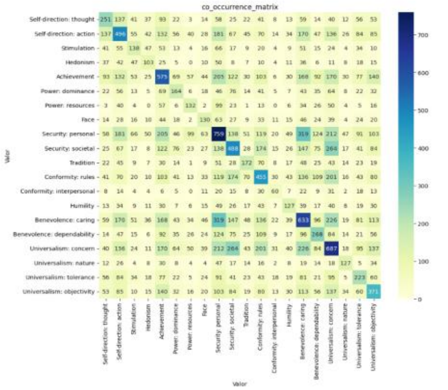


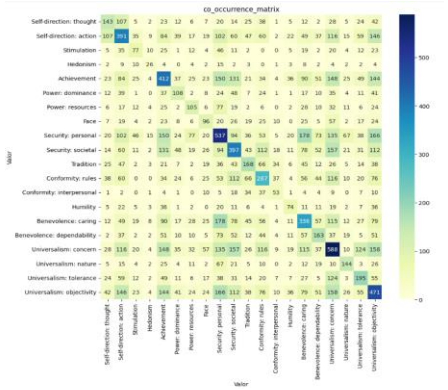




Similarly, the co-occurrence of values exhibits consistent relationships across all three datasets







MODEL

Transformers

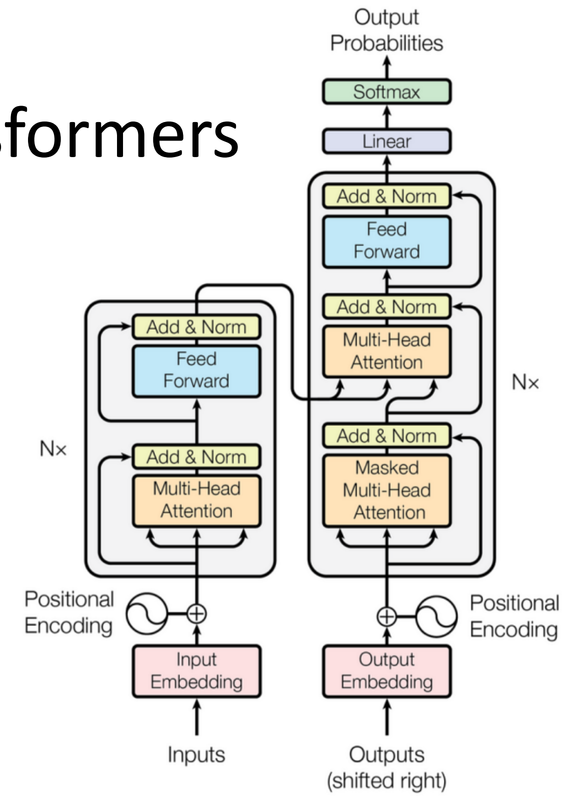


Figure 1: The Transformer - model architecture.

Typically composed of two blocks—an encoder and a decoder—these structures are especially useful for NLP tasks. The most common example of their utility is simultaneous translation, where the model must understand an input sequence and produce a related but potentially different output sequence.

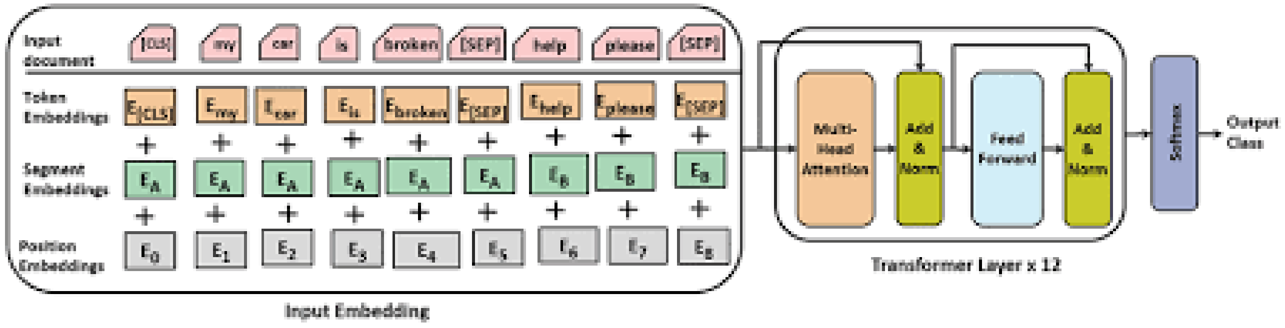
Encoder:
- Contextual Understanding
- Abtraction

Decoder:
- Sequence generation
- Encoded attention

“Attention Is All You Need”, A. Vaswani et al. 2017

MODEL

BERT (Bidirectional Encoder Representations from Transformers)



HUGGING FACE

The BERT Base model consists of 12 layers, each containing a feedforward network with two linear layers and one non-linear activation function, known as 'GELU.' For this specific case, the final layer of the model will be a linear layer with 20 neurons, where each neuron represents a different label.

- Transfer learning
- Cost, Time and Data Efficiency
- Verstility and Broad Language Understanding

METHODOLOGY

HYPERPARAMETER OPTIMIZATION

In the initial phase, we conducted hyperparameter optimization using grid search to assess two specific parameters: Learning Rate and Batch Size. We fixed the number of epochs at 15 and adjusted the data volume due to GPU limitations on Google Colab.

The optimal hyperparameters identified from this search are:

- Learning Rate: 0.00015080099528917156
- Batch Size: 32"

```python
torch.cuda.empty_cache()
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=20)

study = optuna.create_study(direction='minimize')  # Estás minimizando la métrica de la función objetivo
objective_with_data = lambda trial: hyperparameter_search(trial, optimization_dataset)

study.optimize(objective_with_data, n_trials=30)  # Puedes ajustar el número de trials según tus recursos


best_params = study.best_params
print("Best parameters found: ", best_params)
```

```python
def hyperparameter_search(trial, dataset):
    # Configuración del modelo y optimizador

    epochs = 15
    learning_rate = trial.suggest_float('learning_rate', 5e-6, 5e-4)
    batch_size = trial.suggest_categorical('batch_size', [16, 32, 64])
    # Threshold = trial.suggest_categorical('Threshold', [0.33, 0.5, 0.66])

    print(f'This try consider Learning rate = {learning_rate}, batch_size= {batch_size}.')

    model_name = "bert-base-uncased"
    model = BertForSequenceClassification.from_pretrained(model_name, num_labels=20)
    model.to(device)
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

    # # DataLoader (aquí deberías tener tus propios conjuntos de datos)
    train_dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    validation_dataloader = DataLoader(validation_dataset, batch_size=batch_size, shuffle=True)
    # Entrenamiento
    for epoch in range(epochs):
        model.train()
        total_loss = 0.0

        for batch in train_dataloader:
            input_ids, attention_masks, labels = batch
            input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

            outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
            loss = outputs.loss

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

            del input_ids, attention_masks, labels

        average_loss_train = total_loss / len(train_dataloader)
        print(f'Epoch {epoch + 1}/{epochs}, Average Loss: {average_loss_train}')

    total_loss = 0.0
    model.eval()
    y_true = []
    y_pred = []

    for batch in validation_dataloader:
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
            logits = outputs.logits
            total_loss += loss.item()
            predictions = torch.sigmoid(logits)


    average_loss_val = total_loss / len(validation_dataloader)
    print(f'Average Loss: {average_loss_val}')

    return average_loss_val
```

METHODOLOGY

TRAINING

For training, we utilized the complete training dataset and conducted 20 epochs, consistent with the methodology used in J. Kiesel's paper, 'Identifying the Human Values Behind Arguments'.

```
Epoch 1/20, Average Training Loss: 0.4257457693652994
Average Test Loss: 0.3511984246969223
F1 Score Micro: 0.05805927730410069 and F1 Score Macro: 0.022110441185198636
The best epoch is 0 with Loss: 0.3511984246969223
Epoch 2/20, Average Training Loss: 0.346800940276603
Average Test Loss: 0.3215098923444748
F1 Score Micro: 0.30884865049538773 and F1 Score Macro: 0.14488443626554978
The best epoch is 1 with Loss: 0.3215098923444748
Epoch 3/20, Average Training Loss: 0.30578937403549106
Average Test Loss: 0.30917063176631926
F1 Score Micro: 0.3747765317731188 and F1 Score Macro: 0.22089483639026283
The best epoch is 2 with Loss: 0.30917063176631926
Epoch 4/20, Average Training Loss: 0.27139427640734337
Average Test Loss: 0.3070548778772354
F1 Score Micro: 0.422590957527782 and F1 Score Macro: 0.269857528160531
The best epoch is 3 with Loss: 0.3070548778772354
Epoch 5/20, Average Training Loss: 0.24038192180134135
Average Test Loss: 0.3112789511680603
F1 Score Micro: 0.4586226851851851 and F1 Score Macro: 0.2771209217805362
```

The objective of this problem is to minimize loss in multi-label classification. Therefore, I conducted fine-tuning of the BERT base model by adding a final linear layer with 20 nodes to represent each of the task's values. During training, the model was evaluated after each epoch, and the parameters that yielded the best performance were saved

```python
# Entrenamiento
for epoch in range(epochs):
    model.train()
    total_loss = 0.0

    for batch in train_dataloader:
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    average_loss_train = total_loss / len(train_dataloader)
    hist_loss_train.append(average_loss_train)

    print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss: {average_loss_train}')

    model.eval()
    y_true = []
    y_pred = []
    total_loss = 0.0

    for batch in test_dataloader:
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
            logits = outputs.logits
            total_loss += loss.item()
            predictions = torch.sigmoid(logits)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend((predictions > Threshold).cpu().numpy())

    average_loss_val = total_loss / len(test_dataloader)
    hist_loss_test.append(average_loss_val)

    print(f'Average Test Loss: {average_loss_val}')
    f1_micro = f1_score(y_true, y_pred, average='micro')
    f1_macro = f1_score(y_true, y_pred, average='macro')
    recall_micro = recall_score(y_true, y_pred, average='micro')
    recall_macro = recall_score(y_true, y_pred, average='macro')
    print(f'F1 Score Micro: {f1_micro} and F1 Score Macro: {f1_macro}')

    if hist_loss_test and average_loss_val<=min(hist_loss_test) :
        model_path = "/content/drive/MyDrive/Colab Notebooks/INFORET/InfoRet_model.pth"  # Reemplaza con la ruta deseada en tu Google Drive
        torch.save(model.state_dict(), model_path)
        print(f'The best epoch is {epoch} with Loss: {average_loss_val}')
```

METHODOLOGY

TESTING

For evaluating this model, I opted to use the F1-Score-micro, which assesses model performance while considering the significance of each class based on the number of samples present in the dataset.

This metric is particularly relevant to this phase of the project because, initially, I identified a significant imbalance in the sample distribution across the dataset. Therefore, the expected quality of the results is intrinsically linked to the quality of the fine-tuning, which, in turn, is associated with the quality of the data utilized. I anticipate better results for classes with more examples, as the model will have had more opportunity to learn from them.

```python
def evaluation(model, dataset, epochs, learning_rate, batch_size):

    model.eval()

    total_loss = 0.0
    y_true = []
    y_pred = []

    test_dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    for batch in test_dataloader:
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
            logits = outputs.logits
            total_loss += loss.item()
            predictions = torch.sigmoid(logits)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predictions.cpu().numpy())

    average_loss_val = total_loss / len(dataset)
    y_true1 = np.array(y_true)
    y_pred1 = np.array(y_pred)

    return average_loss_val, y_true1, y_pred1

def best_threshold(y_true, y_pred_prob):
    Micro_f1 = []
    Macro_f1 = []
    for i in range(1,101):
        threshold = i/100
        y_pred = y_pred_prob > threshold
        f1_micro = f1_score(y_true, y_pred, average='micro')
        f1_macro = f1_score(y_true, y_pred, average='macro')
        Micro_f1.append(f1_micro)
        Macro_f1.append(f1_macro)
        if f1_micro >= max(Micro_f1):
            best_threshold = threshold
            # print(f'F1 Score Micro: {f1_micro} and F1 Score Macro: {f1_macro} for the Threshold: {threshold}')
        print(f'F1 Score Micro: {f1_micro} and F1 Score Macro: {f1_macro} for the Threshold: {threshold}')

    return best_threshold, Micro_f1, Macro_f1
```

METHODOLOGY

TESTING

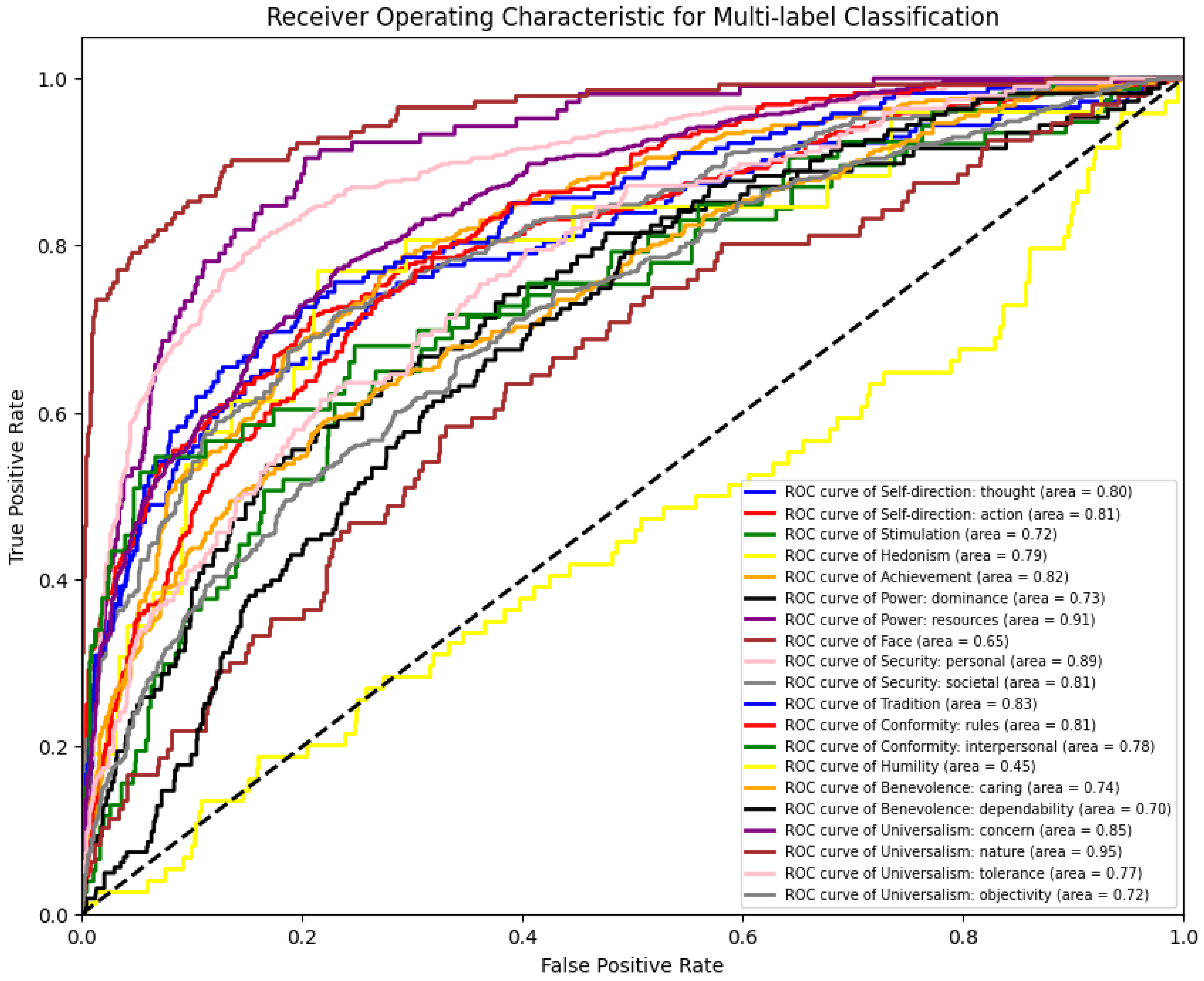
To calculate the F1-Score, it's necessary to select a threshold that determines whether the model's output for each label is considered positive or negative.

Therefore, I have developed a function to identify the optimal threshold based on the F1-Score's performance results.

```
def best_threshold(y_true, y_pred_prob):
    Micro_f1 = []
    Macro_f1 = []
    for i in range(1,101):
        threshold = i/100
        y_pred = y_pred_prob > threshold
        f1_micro = f1_score(y_true, y_pred, average='micro')
        f1_macro = f1_score(y_true, y_pred, average='macro')
        Micro_f1.append(f1_micro)
        Macro_f1.append(f1_macro)
        if f1_micro >= max(Micro_f1):
            best_threshold = threshold
            # print(f'F1 Score Micro: {f1_micro} and F1 Score Macro: {f1_macro} for the Threshold: {threshold}')
        print(f'F1 Score Micro: {f1_micro} and F1 Score Macro: {f1_macro} for the Threshold: {threshold}')

    return best_threshold, Micro_f1, Macro_f1
```

RESULTS

ROC ANALYSIS



From the analysis, it's evident that, on the whole, the model offers superior outcomes compared to a random choice across the majority of labels, affirming its general efficacy. Nonetheless, it's important to highlight that a small subset of labels exhibits significantly lower performance levels. Among these, there is one particular label for which the model's predictive accuracy is notably poor, to the point where its performance is essentially equivalent to, if not worse than, what would be expected from a random classification approach.

F1-SCORE FOR EACH LABEL

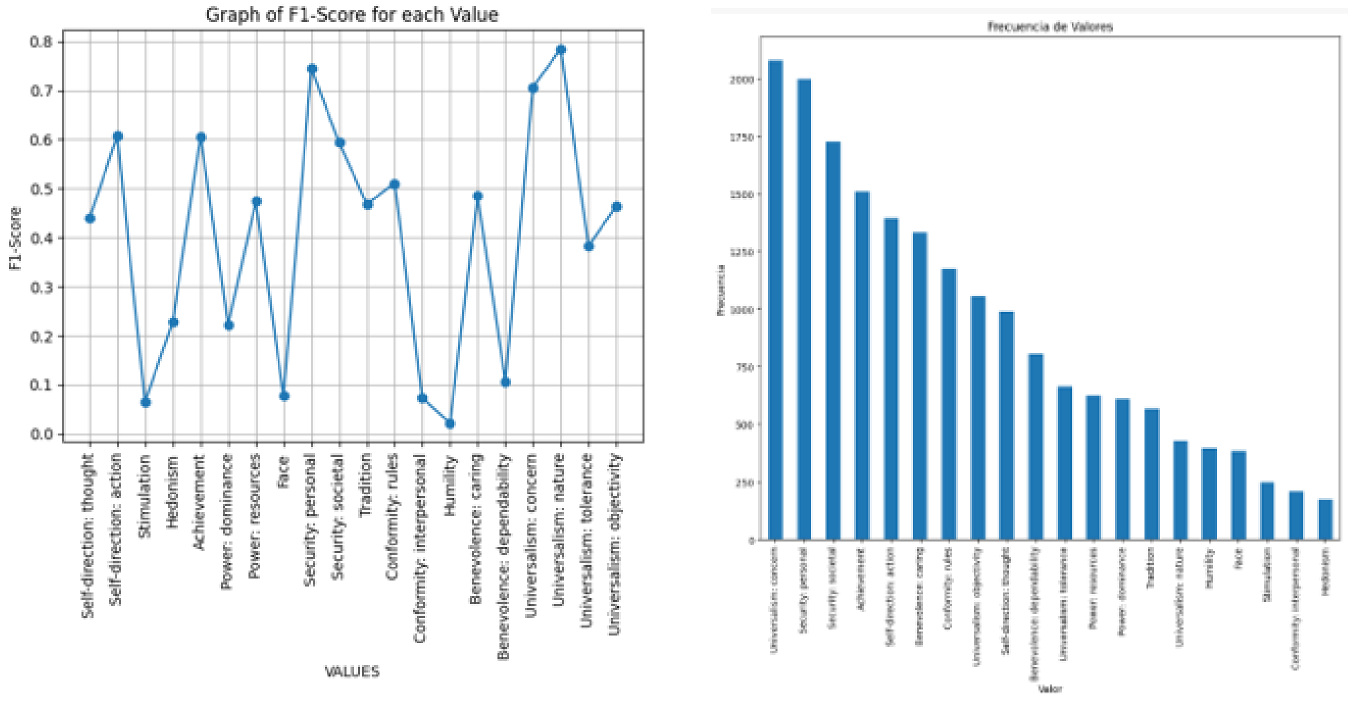


Figure 4: F1-Scores by labels.

However, these outcomes are not entirely unexpected. The classes that exhibit low performance coincide largely with those having a minimal number of samples in the datasets. This observation underscores the fact that the quality of data, in terms of both accuracy and quantity, plays a crucial role in the model's effectiveness. It highlights the importance of a well-balanced dataset for achieving optimal performance across all classes

BENCHMARKING

Results

Download the results for all runs submitted for the deadline as [tab-separated values file].

Main

Best-scoring run of each team on the main test dataset. [all]

TEAM	RUN	F1-SCORE								
		ALL	SELF-DIRECTION: THOUGHT	SELF-DIRECTION: ACTION	STIMULATION	HEDONISM	ACHIEVEMENT	POWER: DOMINANCE	POWER: RESOURCES	FACE
Adam Smith	2023-01-27-17-09-53	0.56	0.59	0.71	0.22	0.29	0.66	0.48	0.52	0.30
John Arthur	2023-01-20-14-38-48	0.55	0.56	0.70	0.27	0.25	0.65	0.50	0.52	0.39
PAI (Theodor Zwinger)	2023-01-27-04-05-18	0.54	0.59	0.71	0.29	0.32	0.61	0.45	0.49	0.36
Mao Zedong	2023-01-26-05-32-20	0.53	0.53	0.70	0.26	0.29	0.60	0.45	0.54	0.31
Confucius	2023-01-20-15-27-39	0.53	0.52	0.71	0.25	0.32	0.61	0.44	0.53	0.39

Given that this project aligns with the challenge 'SemEval 2023 Task 4: ValueEval,' I chose to benchmark my results against those of the challenge participants.

I'm pleased to report that this model achieved a Micro F1-Score of 0.5515, positioning it competitively among the top entries in the challenge. This score underscores the model's robustness in the multi-label classification of human values and highlights the benefits of leveraging pretrained models for this task.

MY LAST CONCLUSIONS

- For this model, the learning rate is crucial in achieving optimal performance.
- The model's rapid convergence concerning the test set results indicates that controlling the learning pace could foster slower learning, leading to better generalization regarding the concept of values.
- Exploring other pretrained models might also yield improved results. While the model utilized here was BERT base-uncased, there are more complex models that could potentially offer a superior ability to generalize these values more effectively.
- The quality of data, in terms of both accuracy and quantity, plays a crucial role in the model's effectiveness.

THANK YOU

MIGUEL CORREA

+39 3497994109

Miguel_correa.g@hotmail.com

INFORMATION RETRIEVAL