SemEval 2023 Task 4. ValueEval: Identification of Human Values behind Arguments

Anonymous ACL submission

Abstract

This paper will detail the methods and model for my submission to SemEval 2023 Task 4, ValueEval: Identification of Human Values behind Arguments. I created a method that used two models, the first predicted whether or not a premise consclusion supported a premise, which was then fed to another model to predict the human emotion behind the argument.

1 Introduction

The SemEval 2023 Task 4 has been explained on their website, but allow me to reiterate the task here. Given a set of premise/conclusions, we are trying to classify the text into 20 categories in a multi-label setting, meaning an argument could exhibit emotions from multiple labels.

The nature of this task is important to the field of natural language processing as we move further and further into a digital future. Having systems that can evaluate the many emotions a speaker is presenting is important from moderation, automation, and several other sub-fields related to software. In terms of moderation, especially in the context of Social Media, being able to flag posts that advocate harm or danger is crucial if we aim to reduce users encouraging others to harm themselves or other people. In terms of automation, any system that listens to a users input and returns a response could be better served by understanding the emotions of the user.

In this paper I will detail the models I built to try and solve this task, including the logic behind the choices I made, the hardware used, and the technologies used.

2 Background

As a current student at the University of Colorado - Boulder, currently enrolled in the Natural Language Processing class, the research and methods I used for this task were entirely formulated from this semester's homework and content. I have been apart of machine learning classes in the past, and for our semester projects we looked beyond our class to find resources to guide our processes, but since I was working alone for this task, I decided I would try something different: I would only use the textbook and material from class.

This definitely is not the "bleeding-edge" approach found in most research tasks, but I really wanted to see what I could come up with using solely my own ideas. This is the first time I have ever attempted a task like this by myself, and it was a lot harder not having the direction of someone else to guide me, and I found that to be an incredibly challenging experience.

3 Experiment setup

3.1 System Design

I approached this problem from two perspectives: a naive model, and a dual model architecture. For both I loaded a BERT pretrained model from one we were given in class ("prajjwal1/bert-small") and one I found in the PyTorch documentation for the sequence classifier ("textattack/bert-base-uncased-yelp-polarity").

In the naive model, the input first went through the layers one might expect an inference model to use. The first layer was passing the tokenized sentences through the BERT layer to get the last layer hidden-state of the first token of the sequence. This was accomplished using HuggingFace's transformer class to instantiate a BERT pre-trained model. These encoded sentences were then passed through a Rectified Linear Unit layer (ReLU), then a Long Short-Term Memory layer (LSTM), which then went to the linear classifier which produced the logits needed for the activation function. The final layer of the first section of layers classified the sentences into two categories, essentially squeezing down into "supports" and "against" (emphasis

on the "naive " in this Naive model). From this prediction, the logits were then included with the BERT-encoded sentences, and passed through the same set of layers to output a label. I did end up modifying the Naive Model a little between the 3-label predictor and the 20-label predictor, namely, the activation function was changed from log softmax to the sigmoid activation function.

If the absolute naivety of this architecture isn't immediately apparent to the reader, allow me to expand. When the forward pass occurs, it is true that the model is predicting a binary label then a multi-class label, however, the loss from the binary classification is mixed in with the loss from the multiclass problem. This means the model can't learn the "stance" of the conclusion to its premise and the emotion of the premise/conclusion pair separately, their losses are being combined into a single value, which is highly detrimental to developing accurate models. This model was intended to be a baseline that could then be compared to the more "advanced" dual model approach.

In the dual model architecture, I trained two separate models, the first predicted if the conclusion supported the premise, and the second predicted which emotion the argument was using to convey its message. I accomplished by training two separate models in the training loop. This allowed me to overcome the shortcomings of the naive model by calculating and propogating the loss from the first model through itself, as well use the logits from the first model as input to the second model. I have seen model architecture that is similar in a way, which is what gave me the idea to attempt this method.

To try and keep the models as close to each other as possible, I used the same number of epochs and learning rate between both the dual and naive model, which was 15 epochs with a learning rate of 5×10^{-6}.

3.2 Hardware

I used a 2021 Apple MacBook Pro with a 32 Gigabytes of RAM on an M1 chip (CPU), as well as a 32-core GPU to run these models. If you're unfamiliar with Apple products, this specific computer uses an ARM architecture CPU (instead of the standard x86 instruction set found on intel chips), which means the instructions must be translated from x86 instructions to ARM instructions, which impacts computation time. Even with this, I was able to achieve ≈ 1 minute train times per epoch on the models trained with "prajjwal1/bert-small", with ≈ 15 seconds needed to evaluate the validation set, and ≈ 5 minutes per epoch with the models trained on

3.3 Technologies

I used a distribution of conda, mini-forge, which is the only way to run Python "natively" on an Apple M1 macbook (Python version 3.9.13). For modeling I exclusively used PyTorch to build the neural network models. I considered using TensorFlow as I have used it to build a question-answering model previously, but stuck with PyTorch as I prefer the way its forward function is defined/used.

For evaluation of the model I used the F1 score function provided by sklearn's metric class. I used this F1 scoring function as it inherently supports multi-label classification, with no modification needed to its source code.

4 Results

I made the assumption that a random model should be able to achieve a $\frac{1}{n}$ F1 score, where n represents the number of labels being predicted, so I aimed to always do at least better than this. I began with predicting only the well-represented labels (higher counts of in-label examples), which were "Self-direction: action", "Security: personal", and "Universalism: concern".

4.1 Naive Model

4.1.1 Validation Set

First we will look at the results from the Naive Model on the Validation set.

Type of Labels	Naive model F1	E[Random F1]
Partial labels (3)	.491	.333
Full labels (20)	.032	.05

Table 1: Results from Naive Model on Validation set

So the model could perform better than random on the small label set, but with multiclass labeling it fell right on its face.

4.1.2 Test Set

Because of the Naive model's poor performance I didn't submit the predictions for the test set to the presenters.

4.2 Dual Model

4.2.1 Validation Set

Type of Labels	Naive model F1	Random F1
Partial labels (3)	.312	.333
Full labels (20)	.016	.05

Table 2: Results from Naive Model on Validation set

Well, I messed with how the layers were constructed, and various different BERT models (BertForSequenceClassification, BertForPreTraining) on the two different models I selected and couldn't get this model to perform better than random. I didn't even submit the predictions to the test set because I didn't need to submit to know I would do worse than the baseline random model.

5 Limitations

I will be the first to admit that tasks such as this, where we are operating in "unexplored" (or at least not yet well recorded) tasks, are best tackled by teams. After attempts to join a team were denied, I boldly made the statement that I could do this task solo. While I was able to get models working, they weren't "good" models by any means, and I imagine my results would be much better with another member on my team.

For example, I also tried making a "Smart" model, that utilized the stance of each premise/conclusion pair instead of trying to predict that label first. I found the BERT Model I believed would have worked (BertForSequence-Classification), but I could not get it to work properly. If I had another person to split the work with, I could've focused on this model while thye worked on the niave model, or vice-versa. My computer being tied up for an hour or more while these models trained is another example of the disadvantages of working solo for tasks like this.

6 Conclusion

In the end, I didn't get a model that could predict on all labels well, but I did figure out how to take the output from one model and feed it to another, which could prove useful in future modeling tasks. I will say I reconfirmed a fact I learned about myself when I was working as a Data Science intern with Seagate Technologies, I am not good at the creative side of machine learning, where the "out-of-the-box" solution doesn't work and you have to create your own.

I do think the "Smart" model would have been able to perform had I gotten it working, but I just didn't have the bandwidth to be debugging and trying to train models at the same time. I will probably continue messing with it until the deadline for the first run of submissions is complete, and my github will be updated accordingly if I figure it out.