Aishwarya Hastak
June 11th, 2024

ML Apprenticeship Take-Home: Sentence Transformers and Multi-Task Learning

Source Code: https://github.com/AishwaryaHastak/SentenceTransformer

Task 1: Sentence Transformer Implementation

Dataset: I used a manually generated dataset for Topic Classification and Sentiment Analysis using ChatGPT.

Model: I decided to use the DistillBERT model since it is smaller model and would work better for this small-scale application. Additionally, BERT is highly adaptable model and can be easily fine tuned to a variety of downstream tasks due to the versatility of its training data. DistillBERT being a smaller version of this versatile model, I think it was a good choice for this use case. I have not made any changes to the base model architecture for this task.

Tokenization: Using the DistillBERT tokenizer provided in the transformers library by HuggingFace, I tokenized the data to a fixed maximum length of 32 tokens.

Embeddings: Using the model from transformer library, I got the embeddings for the tokenized data.

Task 2: Multi-Task Learning Expansion

Task A: Sentence Classification – Classify sentences into classes like News, Politics, Travel, Entertainment, etc.

Task B: Sentiment Analysis – Classify sentences as Positive, Negative or Neutral.

Model Architecture: To support multi-task learning, I added two separate classification heads for each task – topic classification and sentiment analysis. So, the input sentence encodings are passed to the DistillBERT model and we obtain the embeddings. A dropout layer drops 30% of the units from the output embeddings, to introduce regularization. The output CLS embeiddng from the last hidden state is passed through a linear layer that returns a vector of size equal to the number of classes there are to predict.

Training: I have trained the model by alternating between the two tasks. I have used a batch size of 1 for now, the model can later be adapted to handle multiple batches at once. Since the custom dataset is quite small, I have chosen the Training Arguments accordingly.

Aishwarya Hastak
June 11th, 2024

Task 3: Training Considerations

Discuss the implications and advantages of each scenario and explain your rationale as to how the model should be trained given the following:

1. **If the entire network should be frozen.**

   If the entire network is frozen, there won't be any learning since the network weights are fixed. We save on training time and can only dependent on the pre-trained model's capabilities. This would work well when there is a small dataset and the application can rely solely on the pre-trained model's knowledge.

2. **If only the transformer backbone should be frozen.**

   If the transformer weights are fixed and only the weights for the task-specific heads are allowed to learn, then we are essentially using the pre-trained model as is and then fine-tuning to our downstream tasks. In my opinion, this is the best approach, it does not require much knowledge of the model architecture or training time to further train the pre-trained model. The only focus would be to ensure our task specific networks learn.

3. **If only one of the task-specific heads (either for Task A or Task B) should be frozen.**

   In this case, the model would only be able to learn to perform one of the tasks correctly. This could lead to overfitting to a specific task. The other task would rely only on the capabilities of the pretrained model. We could also use this technique to train the model to each task at a time.

Consider a scenario where transfer learning can be beneficial. Explain how you would approach the transfer learning process, including:

Scenario: A chat bot that helps with general knowledge question answering.

1. The choice of a pre-trained model.

   The choice of the pre-trained model depends on many things like my application use case and the compute and memory capacity of my hardware. In NLP, transfer learning on pre-trained models comes in two flavours: representation transfer and parameter transfer [1], which again depends on the use case. There are also metrics that can be used to measure how well a pre-trained model performs during transfer learning [2]. BERT would be a good choice here, since it has been trained on Wikipedia data and is a versatile model overall.

2. The layers you would freeze/unfreeze.

I would freeze the entire or most of the transformer architecture and would only keep the task-specific networks unfrozen and able to learn. Additionally, I would keep all task-specific heads unfrozen, to avoid overfitting to only few or specific tasks.

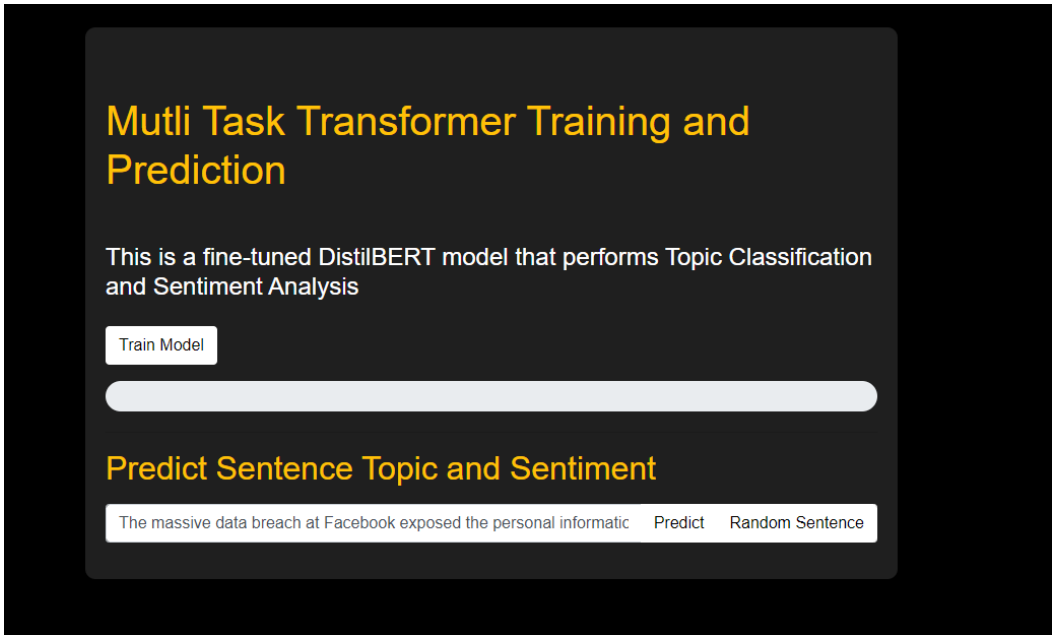
3. The rationale behind these choices.

   Research done by Jaejun Lee et al. [3], found that only the last layers of a pre-trained model need to be fine-tuned to perform well on a specific downstream task. Moreover, only about 25% of the final layers of the PTM need to be fine-tuned to fit to a downstream task. It makes sense not to meddle with the pre-trained model too much, since it holds a lot of abstract and high-level knowledge.

Deployment

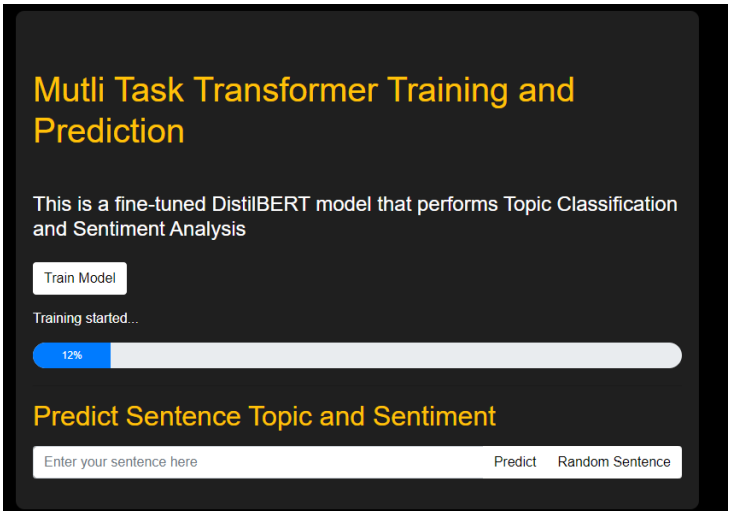
Container: I have packaged the entire application in a Docker file.

Web App: This application can also be locally hosted using the command:
python app.py

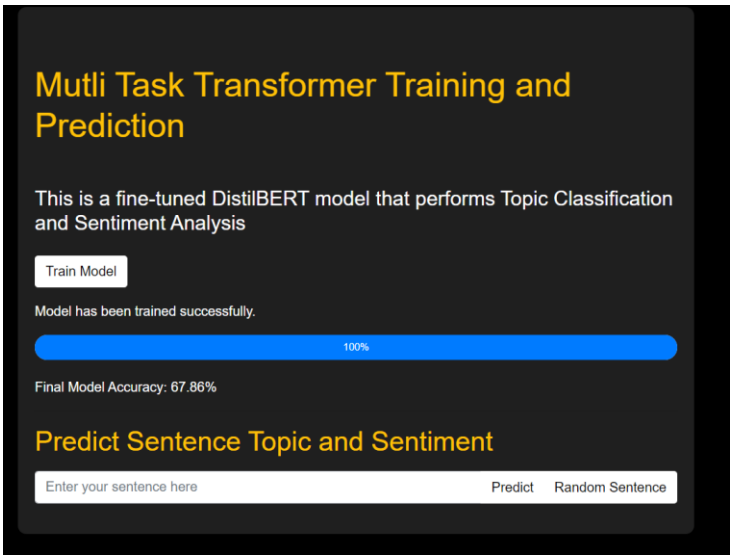


Aishwarya Hastak
June 11th, 2024

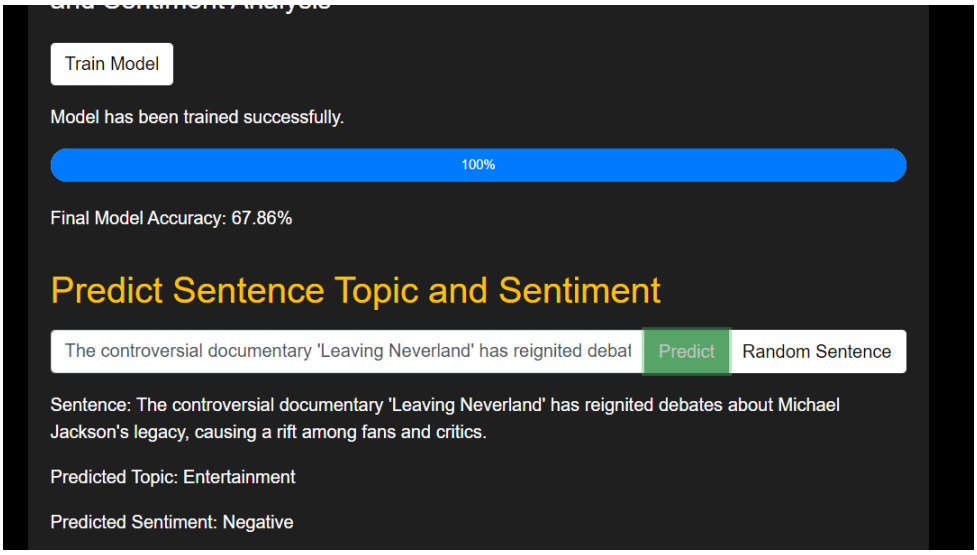
Training the model:



Model trained successfully:



Using Random Sentence and Testing the model:



Aishwarya Hastak
June 11th, 2024

References

[1] Han, Xu, et al. "Pre-trained models: Past, present and future." *AI Open* 2 (2021): 225-250.

[2] You, Kaichao, et al. "Logme: Practical assessment of pre-trained models for transfer learning." *International Conference on Machine Learning*. PMLR, 2021.

[3] Lee, Jaejun, Raphael Tang, and Jimmy Lin. "What would elsa do? freezing layers during transformer fine-tuning." *arXiv preprint arXiv:1911.03090* (2019).