NLP HW1

Ben Akhtar
906526538

November 2, 2023

1 ReadMe

The ReadMe can be found under the filename: README.md in the submission. Additionally, the following is the same replication information as in the ReadMe:

1.1 Instructions For Replication

1. Place the Jupyter Notebook file into google colab or an environment locally that has access to a GPU.
2. If you are running colab, you will need to edit the path to the folder in the first line. Additionally, in the 3rd code block you will need to change every single path to reflect where you first import the csv from, save the three ones created and then re-load them in as datasets. If not running in co-lab, these will need to be local paths, and the necessary commands may need to edited.
3. All other commands should run normally.

2 Working with the Data

For this homework, working with the data required a large amount of extra work. First, we had to import the sentences and tuples along with the relations. From here, it was necessary to split the sentences and tuples to get the label for the current sentence and entity pair. This was then matched to what we had in the relations.txt to give us our label. This was previously converted to a number as it is necessary for the training of our model.

From here, the permutations were made. This required a significant amount of time and work to perform as there were a significant amount of edge cases. For starters, it must be noted that duplicates should not be made. This would cause repeated data that is not necessary and just create more overhead in the training. So, for each sentence, I compared to the next sentence to see if it was the same. If it was I would keep repeating until a different sentence. Once we have all of the same sentence lines, it was necessary to pull all the e1 and e2 and their labels. We would then create all the permutations of these e1 and e2 except for when e1 and e2 would be the exact same. Additionally, there were cases where we had two of the same sentences, with the same e1 and e2 between sentences and a different label. When this is the case, we should not generate two of the same permutations of e2, e1, and a label of 29. Instead, we only generate one. This makes sense so we once again do not generate a duplicate.

From this point we re-save the new dataframe as a csv and re-load it. It is saved so it can be used later on. From here, the labels are encoded to properly get the label. Then, we begin to tokenize. We create a token for the sentence, e1, and e2. This is done by giving Bert the special '[SEP]' character so it can recognize new entities. We tokenize all 3 so Bert has as much info as possible to predict the label. From these inputs, we then get the input ids and attention mask and finally the target using our data label property. Then, we can create some basic parameters for each dataset.

3 Building the Model

Most of the model building was followed using the tutorial found here: Mishra.

The model itself is built off Distilbert, with some customizations. The first reason for using Distilbert is it will take far less time than Bert while providing similar performance in our case. Additionally, the customization is so that we will be able to obtain our final outputs. These will then be compared to the encoded labels so we can predict our F1 score.

Once the model is built, the training loop is created. The scores can be ignored here as the f1 score was not properly calculated. The loop was run for 5 epochs to get a good training score and the ids and mask were fed into the model. From here the output prediction for all 30 classes was generated. Then the max is taken to get our predicted label. This is compared to the target. A similar loop was provided in testing, however, some of the code was cleaned up. Unfortunately, for some reason, the F1 scores do not seem to be right for the test and development set as they are both around .08. The training F1 score is .97. It is possible something in the model or test loop was done incorrectly causing this large of an error.

4 Analysis

The reason Distilbert was chosen, and this was a good tutorial to follow, was that Distilbert runs faster compared to regular BERT, but does maintain a high level of accuracy. This is because there is an overall reduction in the number of layers by about half. Additionally, the token-type embeddings and the pooler have been removed. Considering how long it already took to train this model, running with a full fledged BERT model likely would have been untenable. Also, Bert in general is very good complex tasks such as this due to its bidirectional training. Bert is generally easy to use and handles text analysis such as this extremely well. If we were trying to generate text, Bert would not be the choice.

Other models that were tailored towards high-level text understanding would have also likely done well for this task. They may not have been easily used however, and taken longer to run. Additionally, Bert has a large amount of resources on the web for this task.

5 Evaluation

After training the model and the predict function was used on our model. This achieved the following results:
train: 0.9710465361128658
validation: 0.08835820895522388
test: 0.08050202839756593

These results seem to have some issues with the development and test datasets since there should not be such a large discrepancy. It is likely I have made some error in this regard.

References

Abhishek Kumar Mishra. Fine tuning transformer for multiclass text classification. URL https://colab.research.google.com/github/abhimishra91/transformers-tutorials/blob/master/transformers_multiclass_classification.ipynb#scrollTo=Zcwq13c0NE9c.