CS542 Competition Write Up

Evan Powell and Leland Ling

In approaching the problem presented in the Autocast Competition, we separated the questions by mode and trained separate models for multiple choice, true/false questions, and numeric questions. We trained each of these models in separate notebooks and stored the weights of the models so that it could be loaded into a single notebook, where we made the final predictions on the entire test set and packaged our submission according to the file downloaded from the competition site. Each of these models were trained on a single 8GB memory Nvidia 3070 GPU, and therefore needed data loaders to separate the data into smaller batch sizes to overcome the issue of out of memory issues. The sections below describe what was done to train each of the models.

Multiple Choice

To build the multiple choice model, we utilized the Hugging Face Transformers library and fine-tuned the DistilBertForMultipleChoice model. We chose DistilBert because the model was efficient enough to run in colab but could still be effective after being fine-tuned.

We started by pre-processing the data. This process consisted of creating a prompt for each question. First the background was truncated to 512 minus the number of characters with the question, answer, and two " [SEP] " tokens if the background's length was this number of characters. We also converted the answer labels into their integer representation.

After preprocessing the data, we split it into training and validation sets using a 90:10 ratio. We then tokenized the data using the DistilBertTokenizerFast tokenizer and created a custom PyTorch dataset and dataloader for our multiple choice task. We also wrote custom collate functions to handle the variable number of choices in the dataset.

For fine-tuning the model, we utilized the AdamW optimizer with a learning rate of 5e-5 and weight decay of 0.01. The model was trained for three epochs, and the training and validation losses were monitored. Once the model was fine-tuned, we saved it for later use in making predictions on the test set.

During the evaluation phase, we used the fine-tuned model to make predictions on the test set and recorded the predicted class for each question. The test set was pre-processed in a similar manner as the training data, with each question having tokenized string representations of each of the multiple choice answers. Each of these tokenized answers were then run through the fine-tuned model, generating a probability for each of them. The answer from this selection of probabilities was taken using argmax.

True / False

The True/False model was created, trained, and evaluated using the DistilBERT architecture from Hugging Face's transformers library. First, necessary dependencies were installed, and data was loaded from the provided JSON files. The dataset was filtered to only include true/false questions and preprocessed to create a prompt from the question and background information. Labels were assigned as 0 for "no" and 1 for "yes" answers.

The DistilBERT tokenizer was utilized to tokenize the dataset, and the model was fine-tuned for a sequence classification task with two output labels. The dataset was split into training and validation sets, and the model was trained for 3 epochs using the Trainer class and a specified set of training arguments. The model was then evaluated on the validation set, and its performance was visualized using a confusion matrix.

Next, the model was tested on a separate true/false test set which was preprocessed in the same way as the training set. Predictions were generated for the test dataset using a DataLoader and the fine-tuned model, and the ratio of 1s (yes) predicted by the model was calculated.

Numeric Answer

For the numeric questions, our team made a model using the DistilBertForSequenceClassification architecture from the Hugging Face Transformers library.

To preprocess the data, we first filtered the questions dataset to keep only numeric questions that contained answers. The prompt of each of the questions was generated as follows. First we took the type of answer, float, date-time, and integer. Added a sentence denoting its answer type to the prompt followed by a “ [SEP] “ token. The background and answers were added in a similar manner to the multiple choice model above.

To prepare the data for model training, we used the AutoTokenizer from the Transformers library to tokenize the input strings. We then created a custom PyTorch Dataset class named "numdataset" to store the tokenized inputs and target values.

The model was fine-tuned for three epochs using the AdamW optimizer with a learning rate of 5e-5. We used a batch size of 2 and split the dataset into a 70% training set and a 30% validation set. The loss function used was the default loss function provided by the DistilBertForSequenceClassification model.

After training the model, we saved the trained model to disk for future use. To make predictions, we loaded the test dataset and used a custom predict_single_datapoint function that tokenizes the input text, performs a forward pass through the model, and returns the predicted regression value.

Submission

To create a submission, we used the given ipython notebook, and replaced the random baseline with the three models described above. The questions were evaluated in three batches, one for each type of question. After this, the results were reordered according to the question id. Some of the ids were repeated across several questions, and a character was appended to the second instance to ensure that each answer was submitted with its corresponding question.