Tutorial: Statistical Language Processing (CL III)
Summer 2024

Assignment 4: Due Monday, July 15, 2024 - 17:00

Submission

See the general rules which apply to all assignments. Submit the following files to Moodle (only 1 submission per group):

- Notebook (with .ipynb extension) in which all cells have been run and outputs of all code cells are shown.
- Make sure that the first cell in your notebook contains the honor code with your name(s).
- Fine-tuned model (zipped)

Finetuning a Pretrained Transformer Model

In this assignment, you will finetune a pretrained DistilBERT uncased transformer model for the task of classifying tweets into 4 classes (anger, joy, optimism, sadness). You will use tools from Huggingface Transformers to fine-tune the model on the tweet_eval dataset.

To be able to finetune the model quickly, you need to use a GPU. It is recommended to use a hosted environment such as Google Colab, which offers free access to GPUs.

Before You Start

For this assignment, it is best to use the training loop provided by Huggingface transformers (**Trainer**). It will automatically perform batching for you and use a GPU when it is available.

Read the Huggingface NLP Course tutorial, in particular chapter 3: Fine-tuning a pretrained model.

It is strongly recommended that you complete the Huggingface tutorial that shows how to use the Trainer. You can then adapt it to using DistilBERT as the base model, and using the tweet data for training.

Dataset

The tweet_eval dataset will be used as training data. Use the **emotion** subset of the data, where each tweet is labelled as expressing anger, joy, optimism, or sadness.

Steps

This assignment has a total of 40 points, distributed as follows:

1. Overall Organization/Presentation of the Notebook (Total: 4 pts)

   Your notebook should be well organized and have markdown cells with headers and descriptions of what the code is doing.

2. Notebook Setup (Total: 2 pts)

   - Add a Markdown cell containing the honor code with your name(s)
   - Install and import the packages you need.
   - Mount Google Drive and create two directories, one to save training checkpoints and one to save models
   - Set variable *device*, which you need later:
     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

3. Prepare the Training Data (Total: 5 pts)

   (a) Load the emotion subset of the tweet dataset (1 pt)

   (b) Tokenize the entire dataset (2 pts)

   (c) Create small data splits for cpu training (2 pts)

   For dataset loading instructions, see the "Use this dataset" button in the right column of the HuggingFace page.

   Until you get all bugs out of your code, you should use a cpu:

   Runtime -> Change runtime type -> CPU

   Also, you don't need to use much training data just for debugging your code.

   If device.type == "cpu", use only the first (50/10/10) of the tokenized train, dev, test splits.

4. Pre-Training Tasks (Total: 14 pts)

   - Load the pretrained *distilbert-base-uncased* model (5 pts)

     The training task is to classify a sequence of tokens (tweets) as one of several labels. Therefore, the pretrained model is loaded using *AutoModelForSequenceClassification*.

     When a pretrained model is loaded for fine-tuning, it needs to know some facts about the training data. In addition to the model name, pass the following values when loading the pretrained model:

     - num_labels
       The number of labels in the training data, which can be retrieved from the training data features (don't hardcode!)
     - label2id
       {label:id} dictionary. Retrieve the list of label names from the training data features. Map each label to its index in the list.
     - id2label
       {id:label} dictionary. Reverse mapping of label2id

     Providing the label-id mappings will make it easier later, when you use the model for inference in the last step. The model will return its predictions as strings ("anger", "joy",..). If the mappings are not provided, the model returns predictions as "LABEL_0", "LABEL_1"..., which is inconvenient.

   - Training Arguments (4 pts)

     In addition to the directory for saving training checkpoints (the directory you created when mounting google drive), the following parameters for the training arguments are recommended:

     - save_total_limit=2
       During training, the model is saved at intervals, according to save_strategy. These models are called checkpoints, and can be used to resume training. Since space is limited, and these checkpoints can fill up your google drive rather quickly, we limit their number to 2 (last and best models).
     - load_best_model_at_end=True
       When training is finished, load the best model. Later, when you save the model, you are saving the best model, not the last model.
     - eval_strategy="steps"
     - save_strategy="steps"
     - logging_strategy="steps"
       Evaluate, save, and log every 500 steps (default value for steps is 500). Alternatively, you can use "epoch" as a strategy, just make sure that eval_strategy and save_strategy are the same (or just make them all the same).

   - Evaluation Setup (5 pts)

     Define the compute_metrics function that will be used for evaluation, both during training and for evaluation on the test set. Since this is a classification task, use the F1 score (with macro averaging):

     metric = evaluate.load("f1")

5. Initialize Trainer and Train (Total: 2 pts)

   Important: Move the model to the device before training:

   model.to(device)

   When you are ready to train with the full dataset, request a GPU:

   Runtime -> Change runtime type -> T4 GPU

   Training for the default 3 epochs will take roughly 2 minutes on a GPU.

6. Save the Best Model (Total: 3 pts)

   Save your model to the directory that was created for that purpose when mounting the drive.

   After the model is saved, it is safe to delete the contents of the directory used for training. You will need to do this to avoid running out of space.

   Check often how much of your google drive quota you are using.

   Also keep in mind that items you delete from the file browser on your PC are moved to the trash, where they remain for 30 days before being deleted. This means that they are still counted on your quota.

   Check the contents of the trash and empty often.

7. Load the saved model (Total: 5 pts)

   If all went well, you should now be able to load your model in the same way you loaded the distilbert-base-uncased model earlier, using AutoModelForSequenceClassification. But this time you only need to provide the model location, since no further training is being done.

   Then create a TextClassificationPipeline. Note that the pipeline task is "sentiment-analysis", and you will need to provide the distilbert-base-uncased tokenizer as an argument to the pipeline.

8. Use your model for inference (Total: 5 pts)

   Use the pipeline created in the previous step to classify some tweets (at least 15). You can use tweets from the test set for this part.

   For each tweet, print the model's prediction, the model's confidence score, and the tweet.