The lead TA for this assignment is Risako Owan (owan0002@umn.edu). Please communicate with her via Slack, email, or office hours.

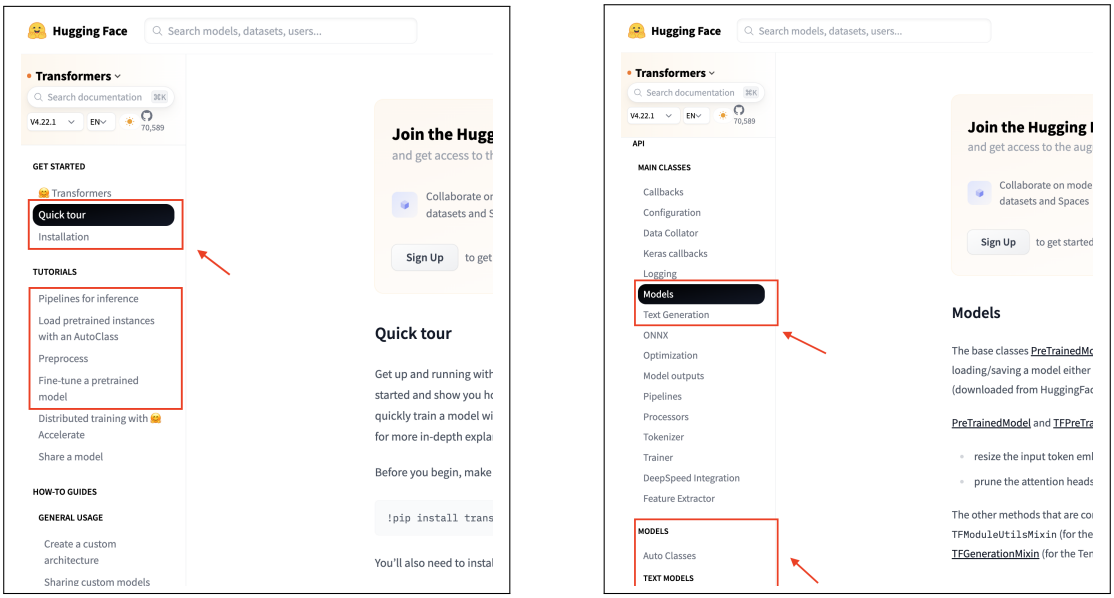
As part of this assignment, you will experience a taste of NLP leaderboard culture and build your own text classifier using the HuggingFace library. The first step is to follow the in-class tutorials on PyTorch and HuggingFace programming to create a text classifier. You may also learn the basic concept of pretraining and finetuning from the HuggingFace tutorial. Also, please make sure you learn how to build a simple neural net-based classifier using a feed-forward neural network in the lecture of text classification on January 25 and 30.

As a next step, you will be replicating the high-performing text classifier on an NLP task at the end of this homework. For the replication, you can choose either (Option 1) using existing code written by authors that appear on the Papers-with-Code leaderboard or (Option 2) fine-tuning the pre-trained model implemented in HuggingFace model libraries. If you don't have much research experience, I highly encourage you to follow Option 2.

If you decide to replicate code from some paper, don't spend too much time making it exactly the same; something similar will suffice. Then, you can jump to Step 2 below and directly replicate the author's code; also, you don't need to use the HuggingFace library if the original code is not written by HuggingFace code. If you decide to fine-tune your classifier using HuggingFace, please follow the steps below.

Note: You will be considered cheating if you do not correctly cite any tools or papers you used. If you have any questions about this, please contact TA Risako Owan.

## Step 1: Getting used to HuggingFace library




Follow the basic instructions on inference, model loading, preprocessing, and fine-tuning in the HuggingFace tutorial: https://huggingface.co/docs/transformers/quicktour. It is highly recommended that you install the library and run the commands in the tutorial in your Google Colab * or your local server using Jupyter Notebook†. In the tutorial document, you can find some default classes imple-

*https://colab.research.google.com/
†https://jupyter.org/

mented by HuggingFace by scrolling down the left menu. You must first understand these abstract classes in order to run their models. A tutorial on fine-tuning HuggingFace's pre-trained model can be found here tutorial.

## Step 2: Choose a Task and Dataset

You can now select a task and dataset from the following list. Please contact TA a week before the deadline if you wish to choose another task and/or dataset. It is recommended that you read the original paper that describes the dataset first. After that, you can download the raw dataset or load it from the pre-formatted HuggingFace dataset. Below are links to the Papers-with-Code leaderboard, original paper, raw dataset, and HuggingFace dataset. Check what model has currently the best score on your dataset in the leaderboard.

Table 1: List of classification tasks and dataset. The following list contains links to the PapersWithCode leaderboard, HuggingFace formatted dataset, and original paper. Question answering (QA) tasks could be viewed as a classification task that predicts the appropriate start and end position of your answer span given a question. Natural Language Inference (NLI) and Human-vs-GPT language detection tasks could be viewed as a classification task as well, as they are predicting the final labels (e.g., entail/contradict/neutral, human/gpt) given a pair of two texts.

| **Tasks** | **Datasets** |
|---|---|
| Sentiment classification | SST2 ( leaderboard, HF dataset, paper )<br>DynaSent ( leaderboard, HF dataset, paper ) |
| Politeness classification | StanfordPoliteness ( dataset, paper ) |
| Social classification | Social Bias Inference (SBIC) ( leaderboard, HF dataset, paper )<br>Hate Speech Detection (HSD) leaderboard, HF dataset, paper ) |
| Natural Language Inference | SNLI ( leaderboard, HF dataset, paper )<br>MNLI ( leaderboard, HF dataset, paper )<br>MRPC ( leaderboard, HF dataset, paper ) |
| Commonsense Reasoning | Winograd Challenge ( leaderboard, HF dataset, paper )<br>CommonsenseQA ( leaderboard, HF dataset, paper ) |
| (Visual) Question Answering | HotpotQA ( leaderboard, HF dataset, paper )<br>SQuAD 2.0 ( leaderboard, HF dataset, paper<br>GQA ( leaderboard, HF dataset, paper )<br>VQA 2.0 ( leaderboard, HF dataset, paper ) |
| Semantic Evaluation (SemEval) | SemEval (2020), SemEval (2021), SemEval (2022),<br>Other SemEval tasks in HF dataset |
| Human-vs-GPT language detection (bonus point) | Human-vs-ChatGPT Comparison Corpus (HC3)<br>( dataset, paper) |

## Step 3: Choose a Model and Replicate it

The final step is to choose a model to replicate. If you choose Option 1, you can start by looking at what models are ranked in the PapersWithCode leaderboards of each dataset and choosing one of the top models (See Figure 1 (left)). If you choose Option 2, you can choose one of HuggingFace's

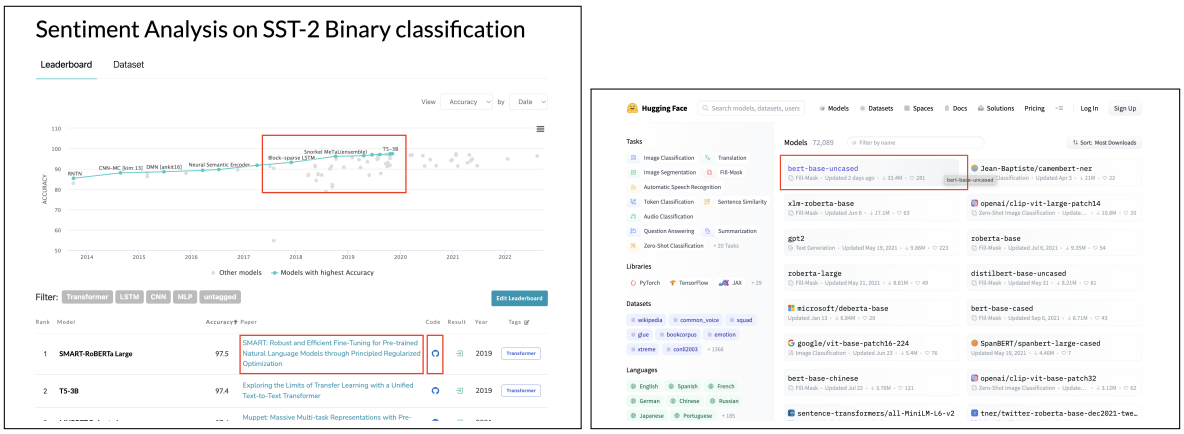



Figure 1: The Papers-with-Code leaderboard of the dataset SST-2 for binary classification task (left) and HuggingFace's model cards on pre-trained language models, like `bert-base-uncased` (right)

pre-trained models, such as BERT, GPT2, or RoBERTa[‡] (See Figure 1 (right)), and fine-tune it. Both cases require you actually "train" the model, either from the code provided by the authors or from writing your own script for fine-tuning the pre-trained model on your target dataset.

If you choose fine-tuning your dataset (Option 2), you are **not** allowed to use the default `Trainer` function in HuggingFace like below.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_imdb["train"],
    eval_dataset=tokenized_imdb["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

Instead, you need to implement your own `Trainer` like `CustomTrainer` and then inherit the default `Trainer` except for `_inner_training_loop` function. You can check how the default `_inner_training_loop` is implemented. In your customized `_inner_training_loop` function, you can just copy the code in the default `_inner_training_loop` function, but please understand how your training process is implemented as discussed in class, such as multiple epochs of training, forward and backward propagation, gradient update methods, gradient clipping, and parameter updating.

From the TA's HuggingFace tutorial, here is an example CustomTrainer.

```
class CustomTrainer(Trainer):
    def _inner_training_loop(
        self, batch_size=None, args=None, resume_from_checkpoint=None, trial=None,
                                                    ignore_keys_for_eval=None
    ):
        number_of_epochs = args.num_train_epochs
        start = time.time()
        train_loss=[]
        train_acc=[]
        eval_acc=[]
```

[‡]I understand you have no idea what BERT/GPT is. We will cover them soon in the class so stay tuned.

```
        criterion = torch.nn.CrossEntropyLoss().to(device)
        self.optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)
        self.scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, 1, gamma=0.9)

        train_dataloader = self.get_train_dataloader()
        eval_dataloader = self.get_eval_dataloader()

        max_steps = math.ceil(args.num_train_epochs * len(train_dataloader))

        for epoch in range(number_of_epochs):
            train_loss_per_epoch = 0
            train_acc_per_epoch = 0
            with tqdm(train_dataloader, unit="batch") as training_epoch:
                training_epoch.set_description(f"Training Epoch {epoch}")
                for step, inputs in enumerate(training_epoch):
                    inputs = inputs.to(device)
                    labels = inputs['labels']

                    # forward pass
                    self.optimizer.zero_grad()
                    # output = ... # TODO Implement by yourself

                    # get the loss
                    # loss = criterion((output[?], labels) # TODO Implement by
                                                           yourself
                    train_loss_per_epoch += loss.item()

                    #calculate gradients
                    loss.backward()
                    #update weights
                    self.optimizer.step()
                    train_acc_per_epoch += (output['logits'].argmax(1) == labels).sum
                                                           ().item()
            # adjust the learning rate
            self.scheduler.step()
            train_loss_per_epoch /= len(train_dataloader)
            train_acc_per_epoch /= (len(train_dataloader)*batch_size)


            eval_loss_per_epoch = 0
            eval_acc_per_epoch = 0
            with tqdm(eval_dataloader, unit="batch") as eval_epoch:
                eval_epoch.set_description(f"Evaluation Epoch {epoch}")
                # ... TODO Implement by yourself
            eval_loss_per_epoch /= (len(eval_dataloader))
            eval_acc_per_epoch /= (len(eval_dataloader)*batch_size)

            print(f'\tTrain Loss: {train_loss_per_epoch:.3f} | Train Acc: {
                                                    train_acc_per_epoch*100:.2f}%')
            print(f'\tEval Loss: {eval_loss_per_epoch:.3f} | Eval Acc: {
                                                    eval_acc_per_epoch*100:.2f}%')

        print(f'Time: {(time.time()-start)/60:.3f} minutes')
```

As part of your assignment or class project, you may have to change some parts of this training function or modify outputs from forward propagation. Your submitted code should include this customized `CustomTrainer` with the copied (or modified) version of `_inner_training_loop` function.

## Step 4: Analyze your classifier's training

Read carefully below what experiments and additional analyses should be included in your report. Missing items will result in point deductions.

- Whether you replicated the classifier from the original author's code (Option 1) or fine-tuned it using HuggingFace (Option 2)
- Description of the task and models with references to the original papers and model cards/repository.
- What kind of hardware you ran your model on
- How do you ensure your model has been trained correctly? Do you have a learning curve graph of your training losses from forward propagation? What does it look like?
- Evaluation metrics used in your experiment
- Test set performance and comparison with score reported in original paper or leaderboard. A justification is needed if it differs from the reported scores.
- Training and inference time
- Hyperparameters used in your experiment (e.g., number of epochs, learning parameter, dropout rate, hidden size of your model) and other details.
- Hypothesize what kinds of samples you might think your model would struggle with and report a minimum of ten incorrectly predicted test samples with their ground-truth labels. If you also report the confidence score of the predicted labels (the last Linear layer's softmax score) on the samples, you will receive a bonus point.
- Potential modeling or representation ideas to improve the errors
- Contribution section - please describe who did what
- (optional) What was the most challenging part of this homework?

## Deliverable

Please upload your code and report to Canvas by Feb 13, 11:59pm.

**Code:** You should provide a zipped file containing your training/inference scripts or a link to your github repository.

**Report:** Maximum six pages PDF.