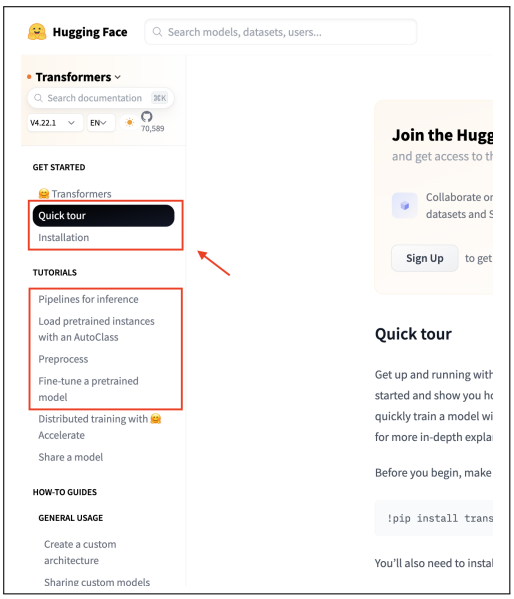
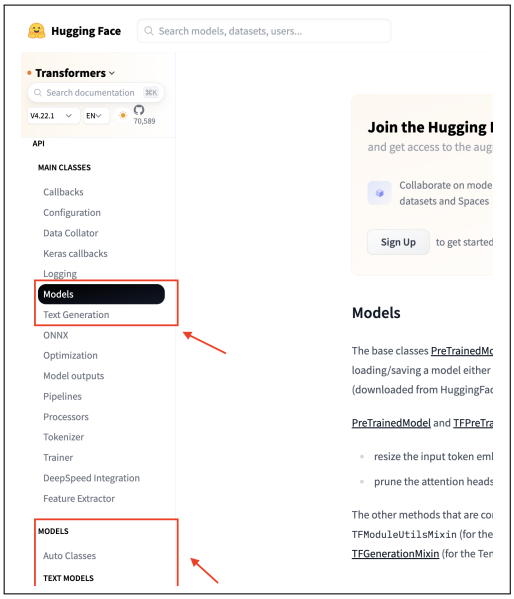
The lead TA for this assignment is Zae Myung Kim (kim01756@umn.edu). Please communicate with the lead TA via Slack, email, or office hours. All questions must be discussed in the homework channel (i.e., #HW1).

**Prerequisite**. This assignment assumes that you have programming experience with PyTorch, Jupyter Notebooks and Google Colab. In case you haven't subscribed to Google Colab Pro, please follow the instructions for subscription and reimbursement at the end of the semester. Also, carefully read tutorial slides on building a text classifier using Scikit-learn and PyTorch taught by TA on Sep 12 and 14, respectively. You may also learn the basic concept of pretraining and finetuning from the lectures (0912 and 0914) and tutorials on HuggingFace.

**Overview**. As part of this assignment, you will build your own text classifier using the HuggingFace library. By fine-tuning the pre-trained model implemented in HuggingFace model libraries on your dataset, you will replicate the high-performing text classifier and evaluate its performance against the state-of-the-art models on the Papers-with-Code leaderboard. You will be considered cheating if you do not properly cite any tools or papers you used or mention any other person you discussed the assignment with. Please contact the lead TA if you have any questions. Please follow the steps below.

## 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 machine using Jupyter Notebook.† In the tutorial document, you can find some default classes implemented 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.

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

## Step 2: Choose a Task and Dataset

You can now select a task and dataset from the list in Table 1. Please contact the lead 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 PapersWith-Code 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. If you choose tasks with *, you will get a bonus point.

| 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* | Human-vs-ChatGPT Comparison Corpus (HC3)<br>( dataset, paper) |

## Step 3: Choose a Model and Replicate it

The next step is to choose a model to replicate. You can (1) choose one of HuggingFace's pre-trained models, such as BERT, GPT2, or RoBERTa[‡] (See Figure 1 (right)), and (2) fine-tune it. You have to "train" the model, by writing your own training script for fine-tuning the pre-trained model on your target dataset. Note: you are **not** allowed to use the default `Trainer` function in HuggingFace like below.

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

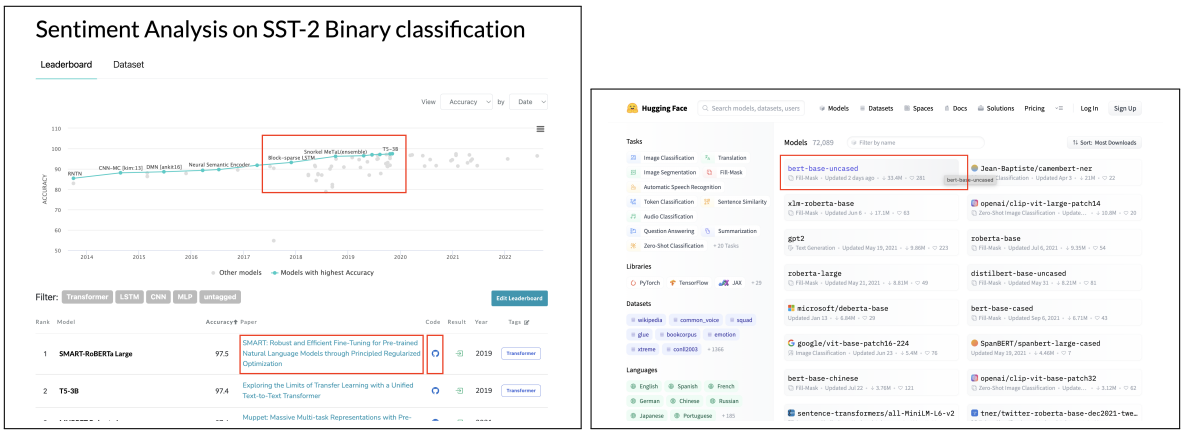



Figure 1: The Papers-with-Code leaderboard of the dataset SST-2 for binary classification task (left) and HuggingFace's model cards on pre-trained language models, like `bert-base-uncased` (right)

```
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=[]

        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 and evaluate it on test set

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

- Description of the task and models with references to the original papers and model cards/repository.
- What kind of hardware you run 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 AND 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?

It is optional to complete the following two steps, but completing them will earn you bonus points.

## (Bonus +2) Step 5: Annotate error types and ideas to fix them

Run your model on the test set and collect incorrectly predicted samples (**no more than 100**[§]) from the test set. If your task has a specific test set from the benchmark, you can use them. You now create a Google spreadsheet and store each error sample in each row with the following information in separate columns:

- Input text
- Ground-truth label (from the original data)
- Predicted label with a confidence score (i.e., softmax output from your classifier with respect to the ground-truth label)

Go through each row and manually label them in the following categories:

- Types of errors, e.g., false positive or false negative
- Types or causes, e.g., over-generalization, surface pattern bias
- Potential solutions to fix the cause, e.g., more training samples[¶], linguistic features, some rules
- Rank your annotations by frequencies and show two tables of distributions of error types and solutions

Figure 2 shows example error annotations with their causes and fixes for the term-definition detection task.[‖] Please note that these types of causes and fixes are specific to the term-definition detection task, so they are not applicable to your task. You should figure out your own error types, causes, and fixes.

[§] If your test set size is smaller than 100, it is fine to report errors less than 100

[¶] Simply labeling most examples with "more training data" without any justification will lose points

[‖] The task that detects spans of scientific terms and definitions defined in text

| Sentence label (gold/pred) (Correct/Wrong) | Result (P/R/F) | Example (gold and predicted token labels) | Error types | Cause (Term) | Cause (Definition) | Surface/parsing patterns | Potential solutions | Difficulty |
|---|---|---|---|---|---|---|---|---|
| none / definition | {'p': 0.2, 'r': 0.134, 'f': 0.161} | We thus utilized Reuters news articles referred to as ' Reuters-21578 , ' which has been widely used in text classification v . We used a prepared SAn exception is the method proposed in ( McCallure and Nigam , 1999 ) , which , instead of labeled texts , uses unlabeled texts , pre - determined categories , and keywords defined by humans for each category .<br>We thus utilized Reuters news articles referred to as ' Reuters-21578 , ' which has been widely used in text classification v . We used a prepared SAn exception is the method proposed in ( McCallure and Nigam , 1999 ) , which , instead of labeled texts , uses unlabeled texts , pre - determined categories , and keywords defined by humans for each category . | False Positive | Over-generalization: technical term bias | Over-generalization (which has \| is the method proposed \| uses): surface pattern bias | | Heuristics: filter out multi-term/definition cases | |

| Error types for term prediction | # | % |
|---|---|---|
| Over-generalization: technical term bias | 26 | 28.9 |
| Missing definition | 13 | 14.4 |
| Incomplete phrase | 11 | 12.2 |
| Wrong data preprocessing | 3 | 3.3 |
| Over-generalization (is a): surface pattern bias | 3 | 3.3 |
| Over-generalization (is): surface pattern bias | 2 | 2.2 |
| Over-generalization (has a): surface pattern bias | 1 | 1.1 |
| Over-generalization (which has \| is the method proposed \| uses): surface pattern bias | 1 | 1.1 |

| Potential solutions to fix errors | # | % |
|---|---|---|
| Heuristics: filter out term/definition only cases | 22 | 20.8 |
| Parse features | 19 | 17.9 |
| Rule: surface patterns | 12 | 11.3 |
| Better encoder | 11 | 10.4 |
| UNK represetnation | 6 | 5.7 |
| Pattern generalization | 6 | 5.7 |
| Annotation: definition vs description | 4 | 3.8 |
| Entity detection | 4 | 3.8 |
| Heuristics: filter out multi-term/definition cases | 3 | 2.8 |
| ? (extremely difficult) | 3 | 2.8 |
| POS features | 3 | 2.8 |
| Heuristics: filter out non-adjacent term:definition pairs | 2 | 1.9 |

Figure 2: Example error annotations (top), example error causes (bottom left), and fixes (bottom right) from the test set for the term-definition detection task. The ground-truth test set has no term and definition annotated, while the model predicts Reuters-21579 and SAn exception as terms, and been widely used in text classification v. and unlabeled texts pre-determined categories as definitions.

## (Bonus +2) Step 6: Visualize errors and perform qualitative analysis

Visualize the errors with other correctly predicted samples (randomly chosen up to 500) in a 2-dimensional semantic space and explore an overall view of how they are projected. Figure 3 shows an example visualization.

**Semantic space**: Take vector representations of correct and incorrect samples from the classifier's output (HuggingFace's model output class). Project them onto reduced dimensions (i.e., 768 dimension $\rightarrow$ 2 dimensions) using dimension reduction methods like PCA (code) or t-SNE [vdMH08] (code). Show the 2-dimensional scatter plot in your report with the observation you found.

When you visualize the scatter plot, please consider the following tips:

- Use Matplotlib (link) or other visualization library for visualization.
- Choose different colors and/or shapes for correct and incorrect samples to distinguish them.
- Use a legend to indicate the type of items
- Display the model's confidence in each sample as transparency using *alpha* variable (example).

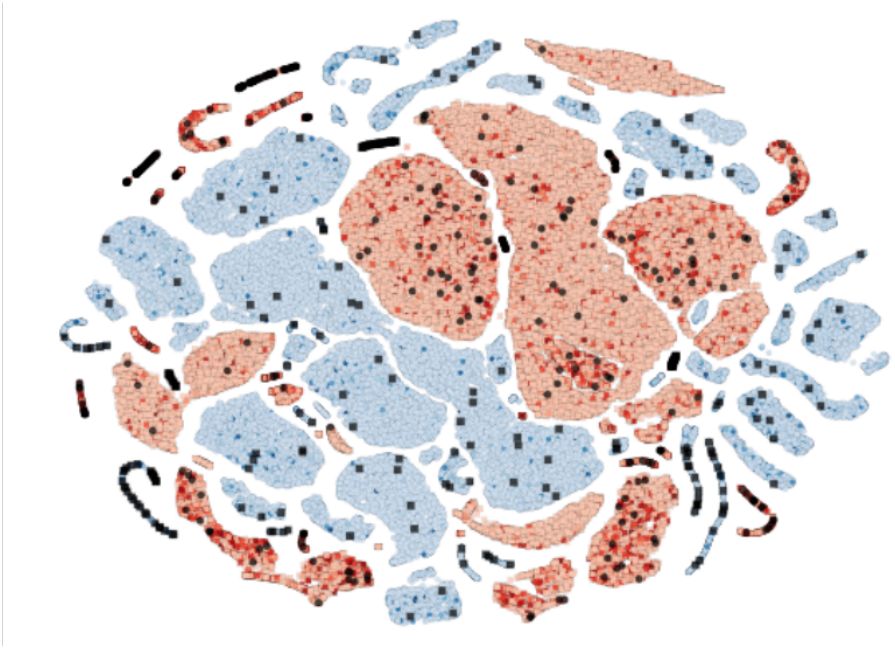


Figure 3: An example t-SNE projection of QNLI dataset: red square and blue circles indicate the QNLI labels whether or not the question is answerable. Incorrectly predicted samples (black) are almost randomly located in the classifier's embedding space.

## Deliverables

Please upload your code and report to Canvas by Oct 1 Sunday, 11:59pm.

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

**Report:** Maximum six pages PDF and other supplementary documents such as spreadsheets for error analysis. The page limit of homework doesn't include references and an appendix with additional information.

## Rubric (15 points)

- Code looks good (+2)
- Specifies which option they chose (+0.5)
- Description of the task and models (+1)
- Includes appropriate references (+1)
- Mentions something about the hardware they used (0.5)
- Explains how they checked their model was trained correctly using learning curve graphs or other appropriate information (+2)
- Specifies evaluation metrics used in experiment (+1)
- Discusses test set performance and comparison with score reported in original paper or leaderboard. Includes justification if it differs from the reported scores. (+2)

- Includes training and inference time (+0.5)
- Includes hyperparameters used in experiment (+1)
- Hypothesis of model performance and/or some kind of discussion about what they found in their incorrectly labeled samples (+1)
- Minimum of ten incorrectly predicted test samples with their ground-truth labels (+1)
- Discusses potential modeling or representation ideas to improve the errors (+1)
- Lists contributions from each member (+0.5)
- Includes confidence scores for incorrectly labeled samples (+1 extra credit)
- Spreadsheet of errors and justification (Step 5: +1 extra credit)
- Spreadsheet of solutions to fix errors (Step 5: +1 extra credit)
- Error Visualizations and Qualitative Analysis (Step 6: +2 extra credit)

## References

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.