The auto-regressive language models (e.g., GPT3 [BMR+20]) trained on human-written text can produce natural text as humans do. In this homework, you will implement various decoding algorithms, generate text using the pre-trained large language models (LLMs) on generation tasks, evaluate the output text, and justify the limitations of current decoding methods.

This assignment assumes that you have covered most of the search algorithms and evaluation metrics in text generation in the Feb 27th lecture on Language Models: Search Algorithms. Please read the reading materials and lecture notes if you missed class. The lead TA for this assignment is Debarati Das (das00015@umn.edu). Please communicate with her via Slack, email, or during office hours.

In this homework, you don't actually need to implement anything from scratch; instead, you will make a complete pipeline of text generation research including decoding, automatic and human evaluation, and analysis of output text. Please follow the steps below and report outputs from the **Tasks** of each step in submitting the spreadsheet, codebase, and report.

Step 1: Implementation of decoding algorithms

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

prompt = "Today I believe we can finally"
input_ids = tokenizer(prompt, return_tensors="pt").input_ids

/* generate up to 30 tokens */
outputs = model.generate(input_ids, do_sample=False, max_length=30)
tokenizer.batch_decode(outputs, skip_special_tokens=True)

/* step 1 */
outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
..

```

You can first go to HuggingFace API on text generation (link) and run an example script to generate text. For instance, once you load pre-trained autoregressive language models like GPT2 [RWC+19], the HuggingFace library allows you to select a variety of decoding algorithms.

You should report the outputs from four different decoding algorithms in this step: *greedy search*, *beam search*, *top-k sampling*, and *top-p sampling* (or nucleus sampling). You don't actually need to implement these algorithms by yourself. Instead, I encourage you to use pre-implemented decoding functions in HuggingFace, like greedy_search(). You will receive a bonus point if you write any of these algorithms from scratch.

Task 1 For a prompt like "Today, I believe we can finally," you should output **output text from the four decoding algorithms with the specific parameters** you used (e.g., beam size, n-best, k, p). For this step, you can choose any random prompt you want. You must also calculate the **perplexity** and the **likelihood of each output sequence** by (log-)summing over every token logit.

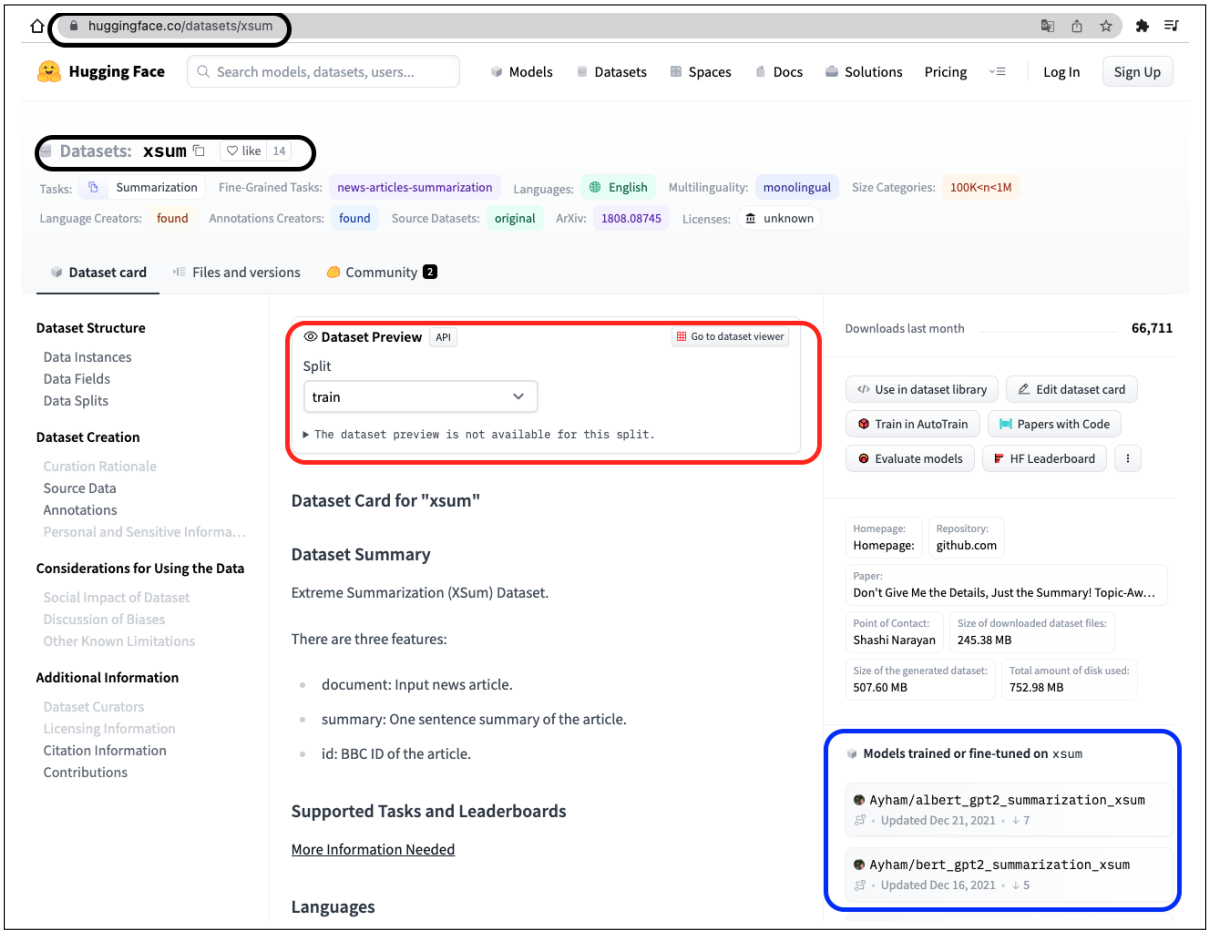

Figure 1: HuggingFace Dataset Interface: https://huggingface.co/datasets?task_categories=task_categories:summarization

Step 2: Decoding for downstream generation tasks

Perplexity is an intrinsic method to evaluate your language model. Now, we evaluate the language model with an extrinsic evaluation. First, select a **specific task** for evaluating decoding algorithms where reference text is provided. For instance, you can choose a dataset called XSUM from the summarization task in the HuggingFace dataset repository, as depicted in Figure 1. You can see some example instances from Dataset Preview and download the dataset easily from a few lines of code:

```
from datasets import load_dataset
dataset = load_dataset("xsum")
```

You are **not allowed** to use the XSUM dataset for your homework. Various tasks are available, such as machine translation, abstraction summarization, dialogue generation, paraphrasing, and style transfer, where the input and output text (reference text) are provided.

Once you choose a specific task and dataset for your text generation, now you need to get actual outputs from the generation model. This homework does not require you to train your own encoder-

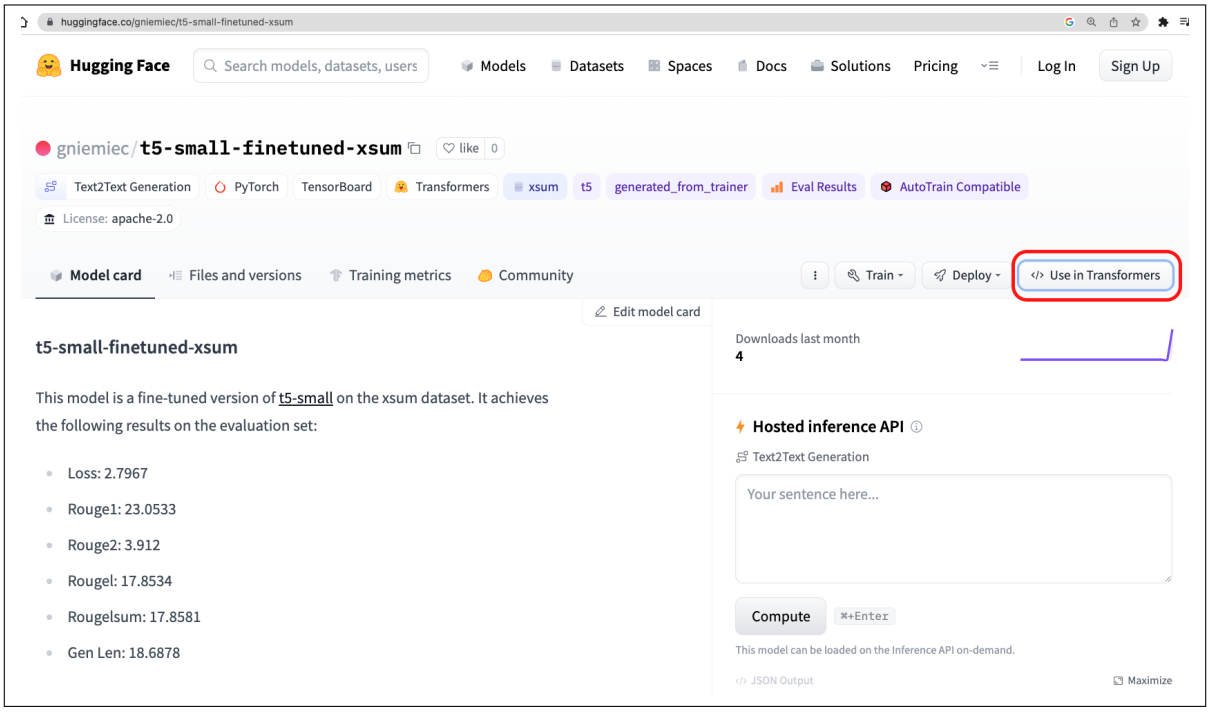

Figure 2: Loading the pre-trained T5 model fine-tuned on XSUM dataset

decoder model from scratch on the target dataset. On the dataset page, you can find the fine-tuned summarization models (blue box in Figure 1).* For instance, you can load the small T5 model already fine-tuned on XSUM dataset https://huggingface.co/gniemiec/t5-small-finetuned-xsum, as shown in Figure 2. On the top right, click the USE IN TRANSFORMERS button to access lines of code for loading the model into HuggingFace.

Task 2 From the test set (or development set if there is no test set given) of the dataset, you can simply generate the output summary. In this step, you have to write your own script to load the fine-tuned model and decode output text given input text from the test samples. Below is an example script to load the model and decode a batch of test input:

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("gniemiec/t5-small-finetuned-xsum")

model = AutoModelForSeq2SeqLM.from_pretrained("gniemiec/t5-small-finetuned-xsum")

/* generate your own summary using different decoding algorithms */
outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
..
tokenizer.batch_decode(outputs1, skip_special_tokens=True)
..
```

*If the dataset page does not include the model list, you can search the dataset name in the model cards.

More specifically, you will use the four decoding algorithms implemented in Step #1 and generate output texts on the test set of your target task. If your test set is more than 50 samples, please only use the first 50 samples in the following evaluation. First, you must make a **spreadsheet** where each row is a text sample in the test set. Then, you need to make four additional columns and copy&paste four output texts from the four decoding algorithms. For instance, your spreadsheet now should be N-by-5, where N is the number of the test set (maximum 50).

Step 3: Automatic and Human Evaluation

Lastly, you must evaluate your generated text by choosing the right evaluation metric for your task. You can find information about the evaluation on text generation in Slides 52-60 of the LM: Search Algorithms lecture. Since this is not an open-ended generation task, you can use your evaluation metrics to determine how similar your generated outputs are to the reference text.[†]

```
1
2  /* generate your own summary using different decoding algorithms */
3  outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
4  outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
5  ..
6  token_outputs1 = tokenizer.batch_decode(outputs1, skip_special_tokens=True)
7  ..
```

Task 3.1 You could choose **at least one** of the content overlap metrics (e.g., BLEU, ROUGE, CIDER, METEOR, PYRAMID) and model-metric metrics (e.g., Word Mover's distance, BERT score), and **measure these metric scores of your decoded outputs with respect to the reference text**. Please consider which automatic metrics would be appropriate for your task and provide a justification in the report.

Again, it is not necessary to implement these metrics from scratch; instead, you can find pre-implemented evaluation metrics at Huggingface's evaluate-metric. As you decode your outputs, you need to report **the metric score of each sample** in a new column in the spreadsheet made in Step #2. The report should include **the averaged metric scores across all samples** (e.g., 50 test samples) and a comparison of the decoding algorithms that work.

Task 3.2 You may notice that automatic evaluation does not accurately measure your task's performance. Here, you will **devise two or three aspects of human evaluation related to your target task**; please check out what types of aspects (e.g., fluency, coherence, formality, typicality) could be used in the human evaluation of generated text in Slides 52-60 of the LM: Search Algorithms lecture. After this, you will **manually annotate their scores with a Likert scale (1-5)** by yourself.

This step aims to identify the gap between automatic evaluation metrics and human evaluations and show their difference in your report. Because human evaluation is time-consuming and costly, you can **only select the first 20 samples** from your test set.

To avoid any biases from the predicted outputs, you should make a new tab in your spreadsheet only with the first 20 test samples and manually label the two or three aspects of text quality in extra columns. Since you are not an expert judge, or your judgment could be subjective, each team member in your team should annotate each sample and aggregate them by majority voting or averaging. Each

[†]I believe you also learned the limitation of single reference-based evaluation as well in the class

team member should have an additional column for their annotation (For example, *factuality_john*, *factuality_jane*, *factuality_joe*). Therefore, if your team members' Likert scores on factuality are 4, 3, and 5, your average score is 4.

Report the averaged automatic and human evaluation scores for the first 20 sentences and describe how they differ in practice.

Deliverable

Please upload your (1) spreadsheet (e.g., CVS, Excel files, Google Spreadsheet) of your decoded outputs with evaluation scores from both automatic and human measurements in Step #2 and #3, (2) zipped code of the decoding algorithms with evaluation script, and (3) technical report (maximum 5 pages) to Canvas by Mar 20, 11:59pm.

Rubric Details

- **Task 1 : Implementation of Decoding Algorithms**
  - Full Marks
  - All 4 decoding algorithms are not implemented: (-2 per algorithm not implemented)
  - Parameters of the algorithms not mentioned (-1)
  - Prompt is not constant across all algorithms (-1)
  - Perplexity not calculated (-1)
  - Likelihood of each output not calculated (-1)
  - No Marks
- **Task 2: Decoding for extrinsic evaluation**
  - Full Marks, the correct implementation of models (loading and decoding) and Nx5 spreadsheet correctly generated
  - XSUM dataset is used (-1)
  - Output summary generated from the train set (-1)
  - Minor mistakes in results or code
  - Major mistakes in results or code
  - No Marks
- **Task 3.1 Automatic Evaluation**
  - Full Marks
  - If only content overlap metrics or only model-metric metrics are implemented (-2)
  - Metrics not calculated between reference text and decoded outputs (-1)
  - Average metric score across all samples not reported (-1)
  - No Marks (no automatic evaluation done)
- **Task 3.2 Human Evaluation**
  - Full Marks
  - At least 2-3 aspects of human evaluation for the target task devised (-1)

  - Reasoning not given behind choice of aspects (-1)
  - Majority/Average voting is not implemented (-1)
  - Difference between human and automatic evaluation is not highlighted (-1)
  - No Marks (no human evaluation done)

- **Report**
  - Full Marks, Report contains full information about models and dataset chosen, the NX5 table and associated metric results, and a comparison of decoding algorithms.
  - Partial, Some pieces of the report are missing
  - No Marks, No report

- **Bonus Point**
  - Implemented decoding algorithm from scratch (+1)
  - Multiple evaluation metrics (2 of each category(content overlap and model-metric) are implemented) (+1)

References

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.