CS336 Assignment #4 (data): Filtering Language Modeling Data

Yin Xiaogang

Spring 2025

If you are using LaTeX, you can use `\ifans{}` to type your solutions.
Please tag the questions correctly on Gradescope, otherwise the TAs will take points off if you don't tag questions.

1. Assignment Review (0 points)

2. Filtering Common Crawl (44 points)

(a) (4 points) Problem (look_at_cc)

i. Download the WARC file above, or find the copy we provide on the cluster. Lets look at the first page in this file. This is a gzipped file, and you can browse its contents with:

```
$ zcat /data/CC/example.warc.gz | less
```

less lets you browse the file using keyboard arrows, Page Up, Page Down. To exit, press q. Look at the very first web page. What is its URL? Is it still accessible? Can you tell what the page seems to be about by looking at the raw HTML?
Deliverable: A 2-3 sentence response.
Solution: I use gzcat command in my macOS(zcat reports an error). Its URL is `http://0371rykj.com/ipfhsb/34.html`. It is not accessible(It is not safe, and is blocked by my chrome. The page seems to be a small company site by looking at the raw HTML.)

ii. Lets now look at the corresponding WET file:

```
$ zcat /data/CC/example.warc.wet.gz | less
```

Note that the WET files contain HTTP headers (e.g., Content-Length) that are not part of the extracted text contents. If you look at the first example, you will see that it contains text that was extracted from the raw HTML you just saw. Notice that much of the extracted text is reminiscent of the HTML structure, and not actually the pages main content. Are there parts of the text you see that you think should have been filtered out by the extractor? Think about the quality of this text as training data: what might go wrong in training a model on text that looks like this? Conversely, what useful information can a model potentially extract from this page?
Deliverable: A 3-4 sentence response.
Solution: Yes, some parts of the text I see should have been filtered out. There are some porngraphic content. Some bad content will be generated by the model which is trained on text that looks like this. In this page, the product introduction can be useful.

iii. What makes a good training example is highly contextual. Describe an application domain for which this example might be useful to have in the training data, and one where it might not be.
Deliverable: A 1-2 sentence response.
Solution: Useful domain: product

iv. Lets look at some more examples to get a better sense of whats in the Common Crawl. Look through 25 more WET records. For each record, very briefly comment on the documents language (if you can identify it), the domain name, what type of page it is, etc. How many examples does it take until you see what youd deem a high-quality webpage?

Deliverable: Brief annotations of 25 documents with the documents language, domain, type of page, and any other miscellaneous notes about the document. The number of examples it takes until you see a high-quality example.

Solution:

No.	language	domain	type
1	Traditional Chinese	product introduction	content page
2	Chinese	movie comment	forum list page
3	English	science	content page
4	Traditional Chinese	porn	content page
5	Traditional Chinese	product introduction	content page
6	Chinese	error page	none page
7	Traditional Chinese	porn	content page
8	Dutch	present	nothing page
9	Greek	education page	content page
10	Greek	forum page	list page
11	Chinese	casino page	content page
12	Turkish	computer tech page	list page(good)
13	English	casino page	content page(good)
14	English	none404 page	bad page
15	Chinese	casino page	content page(good)
16	Traditional Chinese	product introduction	content page(ok http://303323.com/xy
17	Chinese	movie content page	content page
18	Chinese	hospital introduction page	conteng page(good)
19	Traditional Chinese	porn	content page
20	English	search page	none page
21	Traditional Chinese	porn	content page
22	Traditional Chinese	porn	content page
23	Traditional Chinese	porn	content page
24	Spanish	news page	content page(good)
25	Danish	service introduction	content page

it takes about 10 examples until I see a high-quality example.

(b) (3 points) Problem (extract_text)

i. Write a function that extracts text from a byte string containing raw HTML. Use *resiliparse.extract.html2text.extract_plain_text* to perform the extraction. This function needs a string, so you will need to first decode the byte string into a Unicode string. Be aware that the input byte string might not be encoded in UTF-8, so your function should be able to detect the encoding in case UTF-8 fails. Resiliparse also offers *resiliparse.parse.encoding.detect_encoding()*, which might be useful.

Deliverable: A function that takes a byte string containing HTML and returns a string containing the extracted text. Implement the adapter [run_extract_text_from_html_bytes] and make sure it passes *uv run pytest -k test_extract_text_from_html_bytes*

Solution: cs336_data/extract_text.py

ii. Run your text extraction function on a single WARC file. Compare its output to the extracted text in the corresponding WET file. What differences and/or similarities do you notice? Which extraction seems better?

Deliverable: 2-3 sentence response comparing and contrasting the text extracted by your own function versus the extracted text in the WET files.

Solution: Its output has more unwanted character and space. WET's output seems better.

(c) (6 points) Problem (language_identification)

i. Write a function that will take a Unicode string and identify the main language that is present in this string. Your function should return a pair, containing an identifier of the language and a score between 0 and 1 representing its confidence in that prediction.
Deliverable: A function that performs language identification, giving its top language prediction and a score. Implement the adapter [run_identify_language] and make sure it passes both tests in uv run pytest -k test_identify_language . Note that these tests assume a particular string identifier for English (en) and Chinese (zh), so your test adapter should perform any applicable re-mapping, if necessary.
Solution: cs336_data/language_identification.py

ii. The behavior of language models at inference time largely depends on the data they were trained on. As a result, issues in the data filtering pipeline can result in problems downstream. What issues do you think could arise from problems in the language identification procedure? In a higher-stakes scenario (such as when deploying a user-facing product), how would you go about mitigating these issues?
Deliverable: A 2-5 sentence response.
Solution: It will reduce the accuracy of downstream tasks (such as recognizing Chinese as Japanese). In a higher-stakes scenario (such as when deploying a user-facing product), we can raise the threshold or reject some requests or use more advanced models.

iii. Run your language identification system on text extracted from the WARC files (via your previously-implemented text extraction function). Manually identify the language in 20 random examples and compare your labels with the classifier predictions. Report any classifier errors. What fraction of documents are English? Based on your observations, what would be a suitable classifier confidence threshold to use in filtering?
Deliverable: A 2-5 sentence response.

Solution:

id	model_class	score	man_result	right
2313010a-88be-49af-862f-161877b9699e	el	0.999869168	Greek	1
7fe201e8-f0ba-40b3-9dc2-0732347e913d	zh	0.978632092	Chinese	1
eca3f0b3-064e-4c23-ab2d-2701baf06380	ca	0.954317212	Catalan	1
b9cab2be-07d2-4314-a38f-a04a3106fbd0	tr	0.988240182	Turkish	1
5c9cfa3a-0f6c-460a-a69e-7d5489b99bc0	en	0.110449299	English	1
988fe7b4-3dd0-45e2-8ffa-fd06569f75d4	en	0.518493354	English	1
0f7f7694-e93c-4b97-a119-5f9e3a534697	zh	0.872076213	Chinese	1
b43f8a93-7f32-4f22-8578-9f4a0d959447	zh	0.985808253	Chinese	1
2666d698-c60f-49bd-9361-58333a957065	pt	0.927117705	Portugal	1
3ea00414-38db-4ea6-a917-c15370e4099d	en	0.604694903	English	1
e588aaa9-0c8f-4d81-9bf9-8e0cabc9b374	sv	0.815790653	Swedish	1
a1f13415-35ce-4f04-9833-43f6e462f09d	nl	0.909454644	Dutch	1
08a56fb9-1104-49e8-969f-17d1e7795315	en	0.764572859	English	1
b7c3f90f-7448-4a48-a472-9a741f07eac1	de	0.913456917	German	1
dc5aabbb-c532-4f90-830d-c3b0a3fb471a	en	0.30128774	English	1
ed505555-2a41-4eeb-9e59-845fd0c50d43	en	0.363320649	Portugal	0
643fedb7-4304-43cf-9ef4-4979f2d519f7	ru	0.933857203	Russian	1
0168ae62-e9af-4469-8ffa-1ac616184ce9	es	0.981798589	Spanish	1
0105d4a1-3390-410a-92e8-890d82744f8a	en	0.715703964	Chinese	0
91f6840c-9efc-4c27-975b-dbc29a24f5e2	en	0.818909943	English	1

ed505555-2a41-4eeb-9e59-845fd0c50d43 and 0105d4a1-3390-410a-92e8-890d82744f8a are errors. 8/20 are English. Based on my observations, 0.5 would be a suitable classifier confidence threshold to use for languages other than English. For English, the situation is more complex.

English often appears alongside other languages, so we need to use the top 2 score gap value to filter.

3. Constructing Scaling Laws (50 points)

(a) (50 points) Problem (scaling_laws)

Construct a scaling law to accurately predict the optimal model size, its hyperparameters, and the associated training loss for a FLOPs budget of 1e19. To construct your scaling laws, you will use our training API to query the final training loss for various experimental configurations (§3.1); you may not query more than 2e18 FLOPs worth of experiments for fitting your scaling law. This is hard cap that will be enforced by the API.

Deliverable: A typeset write-up that contains a complete description of your approach and methodology for fitting a scaling law. In addition, it should describe how you use the scaling law to predict the optimal model size for the given FLOPs budget, and your predicted values. The write-up should include commentary about why you made particular design decisions, and the description should be detailed enough to reproduce your approach and results.

Note on batch size: We place essentially no constraints on the hyperparameters you may report under the FLOPs budget of 1e19, other than the following requirement: **your batch size must be either 128 or 256.** This is done to ensure that runs have reasonably high model FLOPs utilization. If we have issues with out-of-memory errors when running your reported hyperparameter configuration, we will either use gradient accumulation or scale the number of data parallel GPUs to maintain your desired batch size.

To help you get started, we recommend thinking about at least the following questions. Your writeup should contain additional commentary about how decisions were made for each factor below:

- Given your fixed scaling laws budget of 2e18, how did you decide which runs to query?
- How did you fit your scaling law? Describe the concrete method or methods you used. In particular, it will likely to be useful to familiarize yourself with the approaches used in Kaplan et al.[1] and Hoffmann et al.[2]
- How well does your scaling law fit the experimental data?
- For our given FLOPs budget of 1e19, what optimal model size does your scaling law predict? What is the predicted loss?
- If you were to train a model with your predicted optimal number of parameters, what hyperparameters would you use? To estimate the number of non-embedding parameters for a given model hyperparameter configuration, use $12 n_{layer} d_{model}^2$.

In addition to the report, submit your (1) predicted optimal model size, (2) the training hyperparameters to use including either batch size 128 or 256, and (3) the models training loss to this Google form: `https://forms.gle/sAUSLwCUETew2hYN6`. Part of your grade on the assignment will be determined by the performance of your predicted optimal model.

Solution: I can not access the training api. To be done

References

[1] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. arXiv:2001.08361.

[2] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. arXiv:2203.15556.

Submission Instructions

You shall submit this assignment on GradeScope as two submissions – one for “Assignment 2 [coding]” and another for ‘Assignment 2 [written]”:

1. Run the `collect_submission.sh` script to produce your `assignment2.zip` file.
2. Upload your `assignment2.zip` file to GradeScope to “Assignment 2 [coding]”.
3. Upload your written solutions to GradeScope to “Assignment 2 [written]”.