Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 4003–4012
Marseille, 11–16 May 2020

CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data

Guillaume Wenzek*, Marie-Anne Lachaux*, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, Edouard Grave
Facebook AI
{guw, malachaux, aconneau, vishrav, fguzman, ajoulin, egrave}@fb.com

Abstract

Pre-training text representations have led to significant improvements in many areas of natural language processing. The quality of these models benefits greatly from the size of the pretraining corpora as long as its quality is preserved. In this paper, we describe an automatic pipeline to extract massive high-quality monolingual datasets from Common Crawl for a variety of languages. Our pipeline follows the data processing introduced in fastText (Mikolov et al., 2017; Grave et al., 2018), that deduplicates documents and identifies their language. We augment this pipeline with a filtering step to select documents that are close to high quality corpora like Wikipedia.
Keywords: Common Crawl, web data

1. Introduction

Pre-trained text representations have brought significant performance gains on many natural language processing tasks (Peters et al., 2018). Since the introduction of Transformers (Vaswani et al., 2017) and BERT (Devlin et al., 2018), we have a seen a steady improvement in the quality of these pre-trained models, mainly driven by increasing the size of the pre-training corpora (Radford et al., 2019; Yang et al., 2019; Lan et al., 2019). Nonetheless, the size only does not guarantee better models and the quality of the data has to be preserved, which has lead to the use of ad-hoc datasets created by concatenating existing high-quality data sources like Wikipedia. Unfortunately, such datasets cannot be replicated as easily for low-resources languages, as many have much smaller curated datasets such as Wikipedia.

In this paper, we present a data collection pipeline that allows to gather massive monolingual corpora of high quality in a variety of languages, including many low-resource ones. The principles of our pipeline are general and we show the results of its application to data collected by the Common Crawl project.[1] Common Crawl is a massive non-curated dataset of webpages in many languages, mixed together in temporal snapshots of the web. Our pipeline performs standard document deduplication and language identification similar to Grave et al. (2018), but differs in two ways: first, we preserve the document-level structure to allow for the training of paragraph-level representations like BERT (Devlin et al., 2018) ; second, we add an optional monolingual filtering step that selects documents that are close to high quality sources, like Wikipedia. This is achieved by training a language model on the targeted sources and use the perplexity as a scoring function for documents. Our pipeline can be applied to any number of Common Crawl snapshots and takes 8.5 hours to process per snapshot on 5000 CPU cores. For example, the dataset obtained by preprocessing the February 2019 snapshot is composed of 1.5 billions documents in 174 languages. There are 700 millions filtered documents in English alone, corresponding to 532 billions tokens. That is 120 times bigger than the data used in Devlin et al. (2018).

This paper is organized as follows: we first present the Common Crawl corpora, followed by our overall pipeline to filter high quality documents from it. We then describe additional tools that can be used to tailor the filtering to a targeted corpora. Finally, we give in depth statistics about the dataset obtained from pre-processing a single Common Crawl snapshot. The pipeline and the tools are publicly available[2].

2. Related work

Preprocessing of massive datasets for training text representations has been developed in the context of word embeddings, such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) or fastText (Mikolov et al., 2017). In particular, our pipeline follows the fastText pipeline of Grave et al. (2018) where Common Crawl is split into monolingual datasets using a language identifier based on fastText (Joulin et al., 2016a).

Common Crawl has been used in the context of language modeling to evaluate *n*-gram statistics (Buck et al., 2014). More recently, Baevski et al. (2019) pretrained a BERT-like model on Common Crawl as preprocessed in Grave et al. (2018). In general, progress in sentence representations has been observed by increasing the size of the pre-training corpora (Yang et al., 2019; Liu et al., 2019; Raffel et al., 2019). In particular, and concurrently to our work, Raffel et al. (2019) used a large scale dataset based on Common Crawl to train text representations. Existing work using web based datasets have been using English specific preprocessing, such as keeping URLs shared on Reddit or using hand-crafted filtering rules. As opposed to these approaches, our pipeline can easily be applied to many languages other than English. Closer to this work, Ortiz Suárez et al. (2019) has improved the pipeline of Grave et al. (2018), showing that large monolingual corpora can be extracted from Common

[1]https://commoncrawl.org/about/

[2]github.com/facebookresearch/cc_net

Crawl rapidly even with limited resources. Our work follows a similar pipeline with an additional step to select high-quality documents.

3. Methodology

Every month, Common Crawl releases a snapshot of the web obtained by randomly exploring and sampling URLs. Each webpage is made available different formats: raw (WARC), UTF-8 text (WET), and meta-data (WAT). There is little content overlap between monthly snapshots. The complete archive consists of petabytes of data collected over 8 years of web crawling. The webpages are crawled from the whole web without restriction; they come in many different languages and in the quality of the text varies greatly. The Common Crawl represents a rich resource for monolingual data that comprises a large variety of domains, yet poses challenges due to the large quantity of noisy text.

Here we describe our the methodology used to fetch, deduplicate and filter the Common Crawl data. We focus on preprocessing the text (WET) format of the common crawl snapshots. Our pre-processing pipeline consists of several steps that we describe in this section. An overview of the pipeline is illustrated in figure 1.

3.1. Preprocessing

Each snapshot contain between 20 and 30TB of uncompressed plain text, corresponding to approximately 3 billion web pages (for instance the Feb. 2019 snapshot contains 24TB of data). We download and process each snapshot independently. For each snapshot, we regroup WET files into shards of 5GB each. This makes up for 1600 shards for Feb. 2019 crawl. These shards are saved into a JSON file where one entry corresponds to one web page.

3.2. Deduplication

The first step of our pipeline consists in removing duplicated paragraphs across the different web pages in a snapshot, as they represent 70% of the text. We first normalize each paragraph by lower-casing all characters, replacing numbers by a placeholder (i.e. 0) and removing all Unicode punctuation and accent marks.

Then, the deduplication is done in two independent steps. First, for every shard, we compute a hash code for each paragraph and save them into a binary file. We use the first 64-bits of SHA-1 digits of the normalized paragraphs as the key. Then, we deduplicate every shard by comparing it with either 1, a subset or all of the binary files.

The impact of this choice is discussed in 4. These steps are independent for each shard and can thus be distributed. In addition to removing web copies, this step gets rid of a lot boilerplate such as navigation menus, cookie warnings and contact information. In particular, it removes significant amount of English content from webpages in other languages. This makes the language identification, which is the next step of our pipeline, more robust.

3.3. Language identification

The second step of our pipeline consists in splitting data per language. Following Grave et al. (2018), we use the language classifier from fastText (Joulin et al., 2016b; Grave et al., 2018). The fastText language identifier was trained on Wikipedia, Tatoeba and SETimes. It uses characters n-grams as features, and the hierarchical softmax. It supports 176 languages and outputs a score for each of them in the $[0, 1]$ range. It processes 1k documents per second on a single CPU core. For every web page we compute the most probable language, and the corresponding classifier score. If this score is higher than 0.5, we classify the document in the corresponding language. Otherwise, the language is not clearly identified, and we discard the corresponding page.

3.4. LM filtering

At this step of the pipeline, there are still documents with low quality content. A way to filter out these samples, is to compute a score of similarity of a web page with a targeted domain such as Wikipedia. In this paper, we propose to use the perplexity of a language model trained on the targeted domain as the quality score.

More precisely, for each language, we train a sentence piece tokenizer (Kudo, 2018) and a language model on data from the targeted domain. We use a 5-gram Kneser-Ney model as implemented in the KenLM library (Heafield, 2011) because of its efficiency to process large quantity of data. Then, we tokenize each page in our dataset, with our sentence piece tokenizer and compute the perplexity of each paragraph using our language model. The lower the perplexity, the closer the data is to the targeted domain. At the end of this step, each language is split into three even parts *head*, *middle* and *tail*, corresponding to the perplexity score. In section 5. we show perplexity distributions for one snapshot of Common Crawl.

We have trained sentence piece and Kneser-Ney language models on Wikipedia for 48 languages. We make these models publicly available in the repository. We also provide code to train sentence piece and Kneser-Ney language models and compute the terciles thresholds if the user wants to use other data to filter Common Crawl.

3.5. Reproducing results without the pipeline

Reconstructing the dataset by running our pipeline requires a lot of resources and time. Together with the release of the pipeline, we provide a tool to efficiently reproduce the results of this work. This tool builds on a file containing URLs of webpages and reconstructs the final output of our pipeline from this file.

4. Ablation study

In this section, we discuss the impact of several design choices in our pipeline on the resulting datasets.

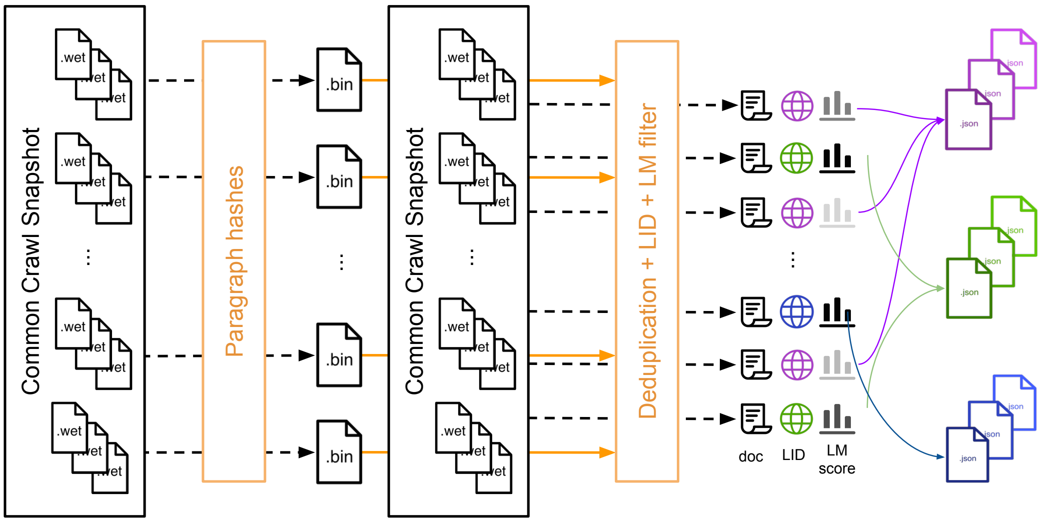

Figure 1: We show the whole pipeline for downloading and processing one snapshot of Common Crawl. First we download all the wet files and compute the paragraph hashes that we group and save into binary files. Then we process every document of the wet files independently: we deduplicate the paragraph using the binary files, we do a language identification and compute language model perplexity score. Finally, we regroup the documents into json files by language and perplexity score. The steps of the pipeline indicated with dashed arrows are parallelisable.

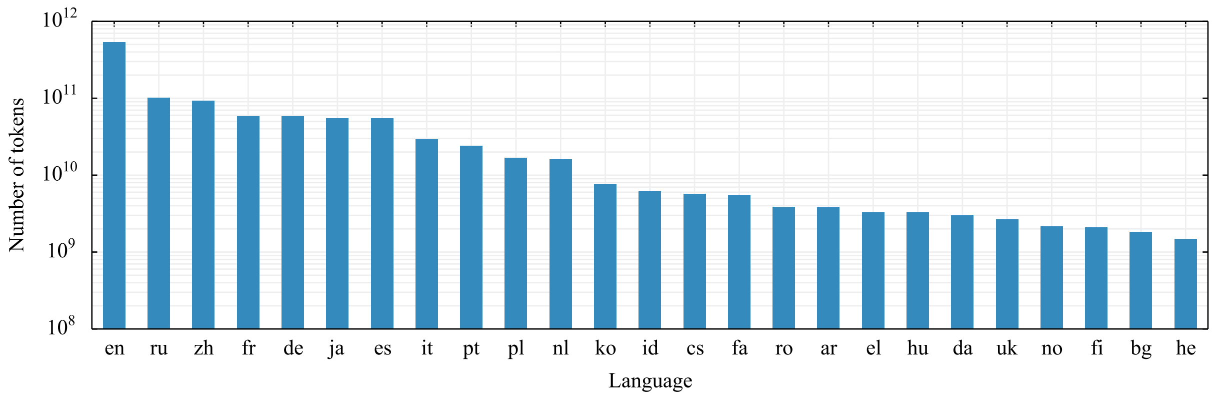

Figure 2: Number of tokens per language for the Feb. 2019 snapshot after deduplication. We display the histogram with logarithmic scale.

4.1. Order of LID and deduplication steps

Contrarily to (Grave et al., 2018), we have chosen to deduplicate the data before language identification, because a lot of English boilerplate, such as cookie warnings, is present in pages of other languages. A significant amount of this noisy data is removed by deduplication which allows for better language identification. This is particularly important for some low resource languages. In Figure 3 we report the relative increase in number of documents when doing ”deduplication then LID” instead of ”LID then deduplication”. We observe that a lot of low resource language documents were mis-classified before deduplication (generally to English), or discarded because no language could be identified.

4.2. Impact of the amount of deduplication

For deduplication, we can compare paragraphs hashes shard by shard, across N shards or across the whole snapshot (1600 shards). The higher N, the higher the number of documents removed and the more RAM the algorithm will use. We show in 4 the amount of data remaining (percentage of number of characters) for one shard of the snapshot Feb. 2019 after deduplication across 1, 2, 5, 10, 20, 50 and 100 shards. After deduplication across 1 shard, there is 42% of characters remaining and 28% across 100 shards. Loading hashes from 50 represents 1.5B unique hashes, making up 13.5GB on disk. Using a memory efficient hashset[3] we can fit those into 40GB of RAM. In 5 we show how the RAM increase when we try to load more hashes in memory. We found 50 shards to be a reasonable

[3]github.com/greg7mdp/parallel-hashmap

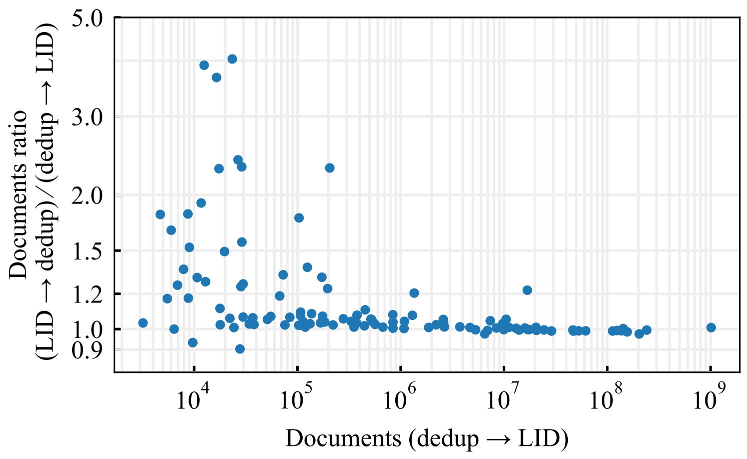

Figure 3: Impact of doing "Deduplication then LID" rather than "LID then Deduplication". Y-axis shows per language-ratio of number of documents between the two methods. X-axis is the number of documents found for each language using LID scores obtained after deduplication. Low resources languages benefits the more from doing "Deduplication then LID" Stats estimated on 1% of Feb. 2019 snapshot.

trade-off and are therefore running the deduplication on blocks corresponding to 3% of the corpus.

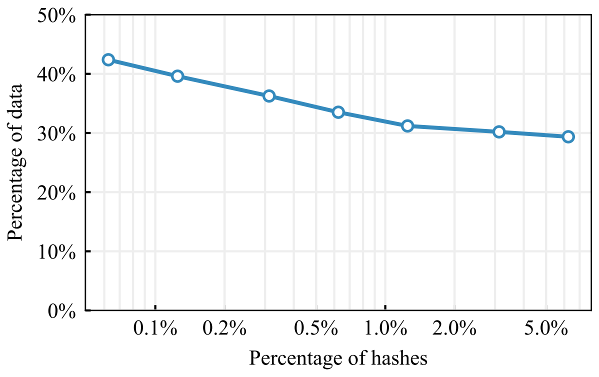

Figure 4: Amount of data remaining after deduplication with different fraction of the dataset. These statistics are computed on one shard.

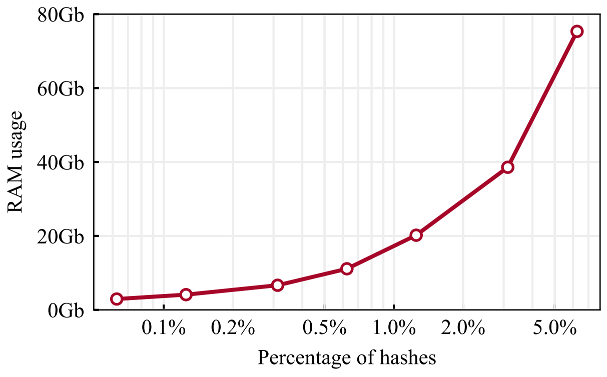

Figure 5: RAM usage when loading hashes from different fraction of the dataset. Computed on one shard.

4.3. Benchmarking

The pipeline is massively parallelizable but still has to run in two steps because of the deduplication which requires to compare billions of documents paragraphs. In our case we chose shards of 5GB as the smallest unit of parallelisation. One dump is divided in 1600 shards, each containing around 1.6M documents. Computing the hashes of paragraphs is done at about 600 doc/s on one CPU core, while downloading the files at the same time. This means that one shard of about 1.6M documents is done in 45 min. We compute all the hashes in 45 minutes on 1600 CPUs. In one pass, the next step removes duplicates, and performs language identification, sentence piece tokenization, language modeling and splitting based on language. Each shard creates 3 files for the top 48 languages for which we have a LM, and one file for each other language where we don't have a LM. Each of those processing require a significant amount of RAM but the memory can be shared across processes since it is read only. This step is significantly longer than the previous one. We allocate 17 processes to one shard. The master process is responsible for downloading the data and distributing the raw documents to the 16 workers as well as writings the results to disk. The worker threads process around 40doc/s, processing the whole shard in about 40 minutes. Removing the duplicated parapgraphs takes 40% of the time. This step is computationally less expensive than the following ones but is done on all the data, as opposed to the next steps which are only applied to the deduplicated data. The language identifier takes 12.5% of CPU time, sentence piece 33% and the LM 13%. Finally we regroup the files produced at the previous steps in chunks of 5Gb. This can be run in parallel for each output file, and since gzip archive can be concatenated without being decompressed first it's very fast and runs in matter of minutes. The total processing time is about 9 hours using 5000 CPU cores for one snapshot.

5. Metrics about the resulting dataset

In this section, we report statistics corresponding to the corpus obtained after applying our pipeline on the Feb. 2019 snapshot of Common Crawl.

5.1. Statistics per language

After preprocessing it, we get 3.2TB of compressed documents in 174 languages. In table 6., we give the sizes of each monolingual corpora for the 130 languages for which we have more than 1000 documents. We also compute the number of tokens and sentences for each language, and report them in Figure 2. The tokens were obtained by using the Sentence Piece tokenizer that was used in our preprocessing pipeline. The sentences were split using Moses. The three largest languages are English (en) with 532B tokens, Russian (ru) with 101B tokens and Chinese (zh) with 92B tokens. We obtained 11 languages with more than 10B tokens, and 27 languages with more than 1B tokens. In terms of documents, the three largest languages are English (en) with 706M documents, Russian (ru) with 167M and German (de) with 105M. There are 12 languages with more than 10M documents and 29 languages containing more than 1M documents. Common Crawl is also a good source for lower resource languages. For example Afrikaans (af), Gujarati (gu), Khmer (km) and Burmese (my) contains respectively 160MB, 190MB, 154MB and 440MB of data. In comparison Wikipedia contains 103MB, 88MB, 71MB and 153MB of data for these languages. And more resources are available through the 60 dumps of Common Crawl. These numbers could probably be improved by increasing the recall of the LID model for low-resource languages.

5.2. Statistics from the language model

We found that perplexity was a relative good proxy for quality. Journalistic and well written content ends up in the head of our dataset. Some documents which contained a lot of keywords list passes through deduplication and LID but receive a high perplexity. Some documents despite being valid text ends up in the tail because they have a vocabulary very different from Wikipedia. This includes blog comments with spoken-like text, or very specialized forums with specific jargon. We decided to not remove content based on the LM score because we think that some of it could be useful for specific applications.
Some languages have very spiked distribution of perplexity while others are more spread out. We postulate that this is rather due to the variance in the Wikipedia sizes used for training the LM than to some language having less high-quality content. Therefore we decided to use different perplexity thresholds for each language. The thresholds have been picked to split the corpus in 3 parts of equal size. In Figure 7 we show the perplexity distribution for two languages English and Gujarati using their respective LM. English LM was trained on 534M of text while Gujarati was trained on only 12M.

5.3. Training models on this dataset

We assess the quality of the resulting dataset by learning unsupervised word and sentence representations through fastText and BERT models. For fastText, we train 300-dimensional word embeddings on the head, middle and tail subsets of the English and Polish CommonCrawl corpora, sorted by document perplexity. We evaluate these on standard semantic and syntactic analogy datasets (Mikolov et al., 2013). We observe in Table 1 a steady increase in performance as we go from the tail to the head of the dataset, confirming the positive impact of our filtering method based on document perplexity.

	English			Polish		
	Total	Sem	Syn	Total	Sem	Syn
head	77.9	81.2	75.3	65.3	66.5	64.1
mid.	74.2	79.0	70.4	62.8	62.7	63.0
tail	62.0	68.1	57.3	59.9	59.8	60.1

Table 1: Impact of corpus quality on the quality of fastText word embeddings. We evaluate on semantic and syntactic similarity datasets.

We also train BERT models on the English (en), Russian (ru), Chinese (zh) and Urdu (ur) languages, using either the Wikipedia corpora or our new CommonCrawl datasets. For these languages, we use respectively 16G, 5G, 1.1G and 106M of raw Wikipedia data (full datasets), and we cap the head CommonCrawl data to 21G, 21G, 17G, 2.2G for English, Russian, Chinese and Urdu. That is, we consider roughly the same amount of data for English, but increase the amount of data for Russian, Chinese and Urdu. We train a BERT-BASE architecture (Devlin et al., 2018) on each of these corpora, without next sentence prediction (NSP) as in (Lample and Conneau, 2019). For better comparison, we early-stop all our models after two days of training on 16 Volta32 GPUs, and use the exact same number of steps for each model. We evaluate each model on the XNLI (Conneau et al., 2018) corpus by using the training data in each language. Results presented in Table 2 indicate that BERT-BASE models trained on CommonCrawl outperform identical models trained on Wikipedia by 3.3% on average. With the same amount of data for English, the BERT-BASE model trained on our corpus outperforms the one trained on the Wikipedia. For low-resource languages like Urdu (ur), the Wikipedia dataset being too small, the model pretrained on Wikipedia obtains similar performance than a randomly initialized model. Using our corpus instead, we obtain a 7 points improvement in accuracy, which demonstrates how our filtered corpus can enable language model pretraining for low-resource languages.

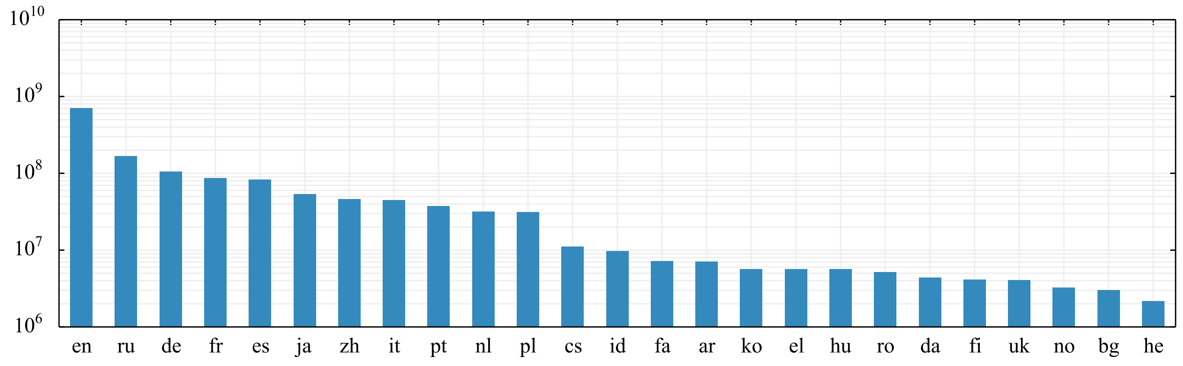

Figure 6: Number of documents per language for the Feb. 2019 snapshot after deduplication. We display the histogram with logarithmic scale. We display statistics for 25 languages only. All statisctics are available in table 6.

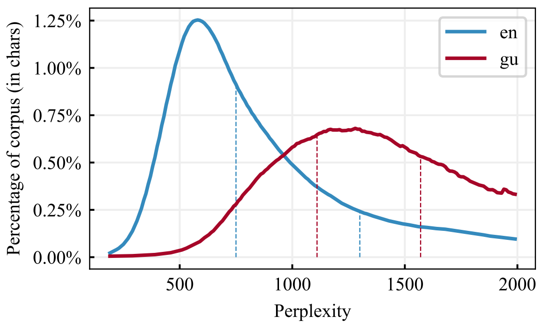

Figure 7: Histogram of language model perplexities for the Feb. 2019 Common Crawl snapshot. The two histograms correspond to English, which is the largest dataset, and Gujarati which is a low-resource language. Vertical lines correspond to perplexity thresholds applied to split the corpus in head/middle/tail.

	en	ru	zh	ur	Δ
Wiki	82.8	73.3	77.0	57.3	72.6
CC	85.0	76.4	77.9	64.3	75.9

Table 2: XNLI dev accuracy for English, Russian, Chinese and Urdu (Δ for average) for BERT-BASE models trained either on Wikipedia or CommonCrawl. The additional data provided by our pipeline alleviates the lack of resources in most languages and enables representation learning for low-resource languages such as Urdu.

6. Conclusion

In this paper, we present a pipeline to create curated monolingual corpora in more than 100 languages. We preprocess Common Crawl by following the pipeline of (Grave et al., 2018), with the differences that we preserve the structure of documents and filter the data based on their distance to Wikipedia. This improves the quality of the resulting dataset and allows for the training of multilingual text level representations like XLM (Lample and Conneau, 2019).

References

Baevski, A., Edunov, S., Liu, Y., Zettlemoyer, L., and Auli, M. (2019). Cloze-driven pretraining of self-attention networks. arXiv preprint arXiv:1903.07785.

Buck, C., Heafield, K., and Van Ooyen, B. (2014). N-gram counts and language models from the common crawl. In LREC, volume 2, page 4. Citeseer.

Conneau, A., Rinott, R., Lample, G., Williams, A., Bowman, S. R., Schwenk, H., and Stoyanov, V. (2018). Xnli: Evaluating cross-lingual sentence representations. In Proc. EMNLP.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning word vectors for 157 languages. arXiv preprint arXiv:1802.06893.

Heafield, K. (2011). KenLM: faster and smaller language model queries. In Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016a). Fasttext.zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016b). Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759.

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. arXiv preprint arXiv:1804.10959.

Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining. arXiv preprint arXiv:1901.07291.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Adv. NIPS.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2017). Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405.

Ortiz Suárez, P. J., Sagot, B., and Romary, L. (2019). Asynchronous pipeline for processing huge corpora on medium to low resource infrastructures. CMLC.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In Proc. EMNLP.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8).

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In Adv. NIPS.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. arXiv preprint arXiv:1906.08237.

Language	Documents	Sentences	Tokens	Size in bytes
af	73.232×10^3	5.390×10^6	73.041×10^6	160.015×10^6
als	7.615×10^3	324.334×10^3	3.526×10^6	9.977×10^6
am	54.182×10^3	1.533×10^6	27.561×10^6	98.878×10^6
an	1.264×10^3	16.818×10^3	213.407×10^3	824.466×10^3
ar	7.132×10^6	248.711×10^6	3.777×10^9	13.602×10^9
arz	44.384×10^3	379.209×10^3	4.222×10^6	18.821×10^6
as	12.758×10^3	563.956×10^3	10.463×10^6	24.174×10^6
ast	3.814×10^3	104.890×10^3	1.357×10^6	3.907×10^6
az	507.612×10^3	21.341×10^6	232.269×10^6	757.804×10^6
azb	12.733×10^3	306.696×10^3	4.365×10^6	13.829×10^6
ba	30.195×10^3	1.047×10^6	12.923×10^6	44.612×10^6
be	176.037×10^3	9.719×10^6	124.716×10^6	476.612×10^6
bg	3.002×10^6	129.758×10^6	1.835×10^9	6.224×10^9
bn	941.621×10^3	38.413×10^6	708.464×10^6	1.717×10^9
bo	30.028×10^3	528.293×10^3	47.940×10^6	50.786×10^6
bpy	2.514×10^3	41.791×10^3	549.568×10^3	1.536×10^6
br	21.594×10^3	653.440×10^3	7.413×10^6	17.989×10^6
bs	12.906×10^3	72.042×10^3	506.201×10^3	3.476×10^6
ca	2.018×10^6	70.986×10^6	1.230×10^9	2.841×10^9
ce	5.534×10^3	222.896×10^3	2.401×10^6	7.812×10^6
ceb	58.489×10^3	2.044×10^6	11.655×10^6	26.459×10^6
ckb	116.103×10^3	2.696×10^6	62.219×10^6	146.949×10^6
cs	11.140×10^6	444.808×10^6	5.691×10^9	17.306×10^9
cv	13.312×10^3	392.207×10^3	3.840×10^6	14.483×10^6
cy	127.800×10^3	4.249×10^6	56.984×10^6	132.824×10^6
da	4.411×10^6	209.623×10^6	2.974×10^9	7.100×10^9
de	105.425×10^6	4.249×10^9	58.195×10^9	164.540×10^9
dv	26.274×10^3	841.155×10^3	9.899×10^6	35.401×10^6
el	5.681×10^6	201.470×10^6	3.282×10^9	12.174×10^9
en	706.583×10^6	32.110×10^9	532.368×10^9	1.955×10^{12}
eo	126.188×10^3	6.152×10^6	70.107×10^6	161.092×10^6
es	82.991×10^6	3.048×10^9	54.792×10^9	134.206×10^9
et	1.043×10^6	56.678×10^6	537.668×10^6	1.598×10^9
eu	381.323×10^3	10.355×10^6	109.635×10^6	342.909×10^6
fa	7.201×10^6	282.130×10^6	5.441×10^9	15.571×10^9
fi	4.118×10^6	191.905×10^6	2.089×10^9	6.836×10^9
fr	86.176×10^6	3.540×10^9	58.428×10^9	220.869×10^9
fy	31.228×10^3	1.087×10^6	12.082×10^6	31.296×10^6
ga	59.515×10^3	2.068×10^6	29.632×10^6	73.301×10^6
gd	10.114×10^3	225.829×10^3	4.132×10^6	9.906×10^6
gl	400.289×10^3	12.171×10^6	196.539×10^6	487.379×10^6
gu	98.263×10^3	4.705×10^6	71.586×10^6	189.806×10^6
he	2.166×10^6	124.089×10^6	1.470×10^9	4.583×10^9
hi	1.370×10^6	52.221×10^6	1.165×10^9	2.762×10^9
hr	821.782×10^3	40.070×10^6	515.230×10^6	1.413×10^9
hsb	8.914×10^3	216.630×10^3	2.288×10^6	8.500×10^6
hu	5.643×10^6	249.899×10^6	3.272×10^9	10.232×10^9
hy	308.674×10^3	10.995×10^6	152.337×10^6	579.637×10^6
ia	1.460×10^3	17.315×10^3	291.786×10^3	930.327×10^3
id	9.728×10^6	488.888×10^6	6.124×10^9	15.782×10^9
ilo	3.990×10^3	131.515×10^3	1.671×10^6	4.421×10^6
io	1.051×10^3	22.527×10^3	174.627×10^3	653.994×10^3
is	346.180×10^3	13.072×10^6	173.198×10^6	502.002×10^6
it	45.080×10^6	1.637×10^9	29.381×10^9	72.517×10^9
ja	53.880×10^6	4.092×10^9	54.883×10^9	127.792×10^9
jbo	1.261×10^3	171.615×10^3	1.514×10^6	1.873×10^6
jv	2.165×10^3	358.813×10^3	5.185×10^6	11.502×10^6
ka	368.404×10^3	16.747×10^6	176.632×10^6	695.075×10^6

kk	208.652×10^3	11.658×10^6	134.347×10^6	526.160×10^6
km	85.211×10^3	2.103×10^6	87.503×10^6	153.530×10^6
kn	112.553×10^3	5.733×10^6	95.568×10^6	217.285×10^6
ko	5.707×10^6	361.022×10^6	7.590×10^9	11.969×10^9
krc	1.696×10^3	92.260×10^3	926.524×10^3	3.371×10^6
ku	49.678×10^3	1.843×10^6	24.831×10^6	61.128×10^6
kv	1.003×10^3	56.540×10^3	586.116×10^3	2.008×10^6
ky	92.894×10^3	2.988×10^6	36.216×10^6	131.434×10^6
la	75.987×10^3	2.932×10^6	41.604×10^6	101.977×10^6
lb	30.740×10^3	965.947×10^3	11.306×10^6	32.277×10^6
lez	1.735×10^3	95.626×10^3	987.272×10^3	3.274×10^6
lmo	1.219×10^3	19.033×10^3	243.748×10^3	798.331×10^3
lo	44.895×10^3	903.215×10^3	22.707×10^6	51.361×10^6
lt	1.485×10^6	63.860×10^6	780.747×10^6	2.337×10^9
lv	846.034×10^3	33.904×10^6	440.403×10^6	1.336×10^9
mg	14.670×10^3	409.271×10^3	6.172×10^6	14.535×10^6
mhr	4.091×10^3	114.556×10^3	1.215×10^6	4.699×10^6
mk	268.409×10^3	8.653×10^6	136.563×10^6	482.894×10^6
ml	292.062×10^3	13.485×10^6	327.757×10^6	559.466×10^6
mn	161.780×10^3	6.790×10^6	90.969×10^6	339.398×10^6
mr	151.850×10^3	8.132×10^6	130.610×10^6	319.199×10^6
ms	373.244×10^3	6.964×10^6	85.562×10^6	259.173×10^6
mt	27.734×10^3	1.096×10^6	17.246×10^6	44.436×10^6
my	170.775×10^3	6.712×10^6	79.498×10^6	439.538×10^6
mzn	2.483×10^3	16.401×10^3	146.797×10^3	1.085×10^6
nds	16.518×10^3	380.501×10^3	4.051×10^6	11.791×10^6
ne	184.598×10^3	6.278×10^6	147.027×10^6	333.808×10^6
new	3.670×10^3	68.984×10^3	1.632×10^6	3.705×10^6
nl	31.635×10^6	1.214×10^9	15.946×10^9	41.821×10^9
nn	123.371×10^3	4.705×10^6	55.776×10^6	145.495×10^6
no	3.268×10^6	158.837×10^6	2.145×10^9	5.524×10^9
oc	9.138×10^3	300.022×10^3	4.327×10^6	10.648×10^6
or	65.718×10^3	961.342×10^3	33.005×10^6	73.860×10^6
os	3.723×10^3	163.153×10^3	1.762×10^6	5.828×10^6
pl	31.242×10^6	1.300×10^9	16.661×10^9	49.738×10^9
pms	4.087×10^3	72.314×10^3	1.124×10^6	2.262×10^6
pnb	12.195×10^3	221.196×10^3	2.752×10^6	8.905×10^6
ps	69.971×10^3	1.975×10^6	43.603×10^6	109.935×10^6
pt	37.305×10^6	1.489×10^9	23.875×10^9	57.388×10^9
ro	5.187×10^6	222.040×10^6	3.848×10^9	9.904×10^9
ru	167.323×10^6	7.718×10^9	101.143×10^9	384.733×10^9
sa	10.064×10^3	794.837×10^3	19.843×10^6	32.559×10^6
sah	8.403×10^3	434.283×10^3	4.135×10^6	14.271×10^6
sd	31.636×10^3	1.133×10^6	22.065×10^6	53.052×10^6
sh	66.385×10^3	2.569×10^6	8.072×10^6	27.332×10^6
si	154.658×10^3	7.072×10^6	124.514×10^6	270.364×10^6
sk	4.472×10^6	115.211×10^6	1.618×10^9	4.787×10^9
sl	1.828×10^6	50.734×10^6	749.341×10^6	2.101×10^9
sq	687.411×10^3	23.223×10^6	392.871×10^6	929.548×10^6
sr	1.344×10^6	56.660×10^6	717.548×10^6	2.108×10^9
sv	15.774×10^6	479.216×10^6	7.149×10^9	19.160×10^9
sw	66.205×10^3	1.915×10^6	36.508×10^6	84.468×10^6
ta	944.262×10^3	48.390×10^6	1.002×10^9	1.513×10^9
te	324.091×10^3	13.951×10^6	225.516×10^6	491.376×10^6
tg	95.142×10^3	3.524×10^6	52.462×10^6	167.373×10^6
th	6.639×10^6	181.397×10^6	2.743×10^9	7.869×10^9
tk	10.841×10^3	347.561×10^3	4.722×10^6	14.295×10^6
tl	192.164×10^3	12.370×10^6	154.572×10^6	329.472×10^6
tr	19.454×10^6	478.459×10^6	6.427×10^9	20.045×10^9
tt	112.660×10^3	3.721×10^6	46.220×10^6	158.642×10^6

ug	27.041×10^3	803.821×10^3	13.479×10^6	44.824×10^6
uk	4.100×10^6	199.198×10^6	2.672×10^9	9.877×10^9
ur	506.610×10^3	8.579×10^6	289.277×10^6	745.210×10^6
uz	35.274×10^3	1.242×10^6	17.024×10^6	45.010×10^6
vi	16.207×10^6	529.567×10^6	9.836×10^9	20.272×10^9
vo	4.934×10^3	72.943×10^3	629.462×10^3	1.689×10^6
wa	1.548×10^3	77.610×10^3	758.117×10^3	1.735×10^6
war	14.530×10^3	162.150×10^3	1.160×10^6	4.310×10^6
wuu	2.907×10^3	4.252×10^3	70.935×10^3	956.106×10^3
xmf	3.854×10^3	116.818×10^3	764.346×10^3	3.128×10^6
yi	30.177×10^3	1.630×10^6	23.397×10^6	59.623×10^6
zh	46.264×10^6	3.081×10^9	92.373×10^9	140.366×10^9

Table 3: Number of documents, sentences and tokens after deduplication.