Common Crawl - Lorelei Language Classification

Outline

1. Problem presentation
2. Architecture
3. Datasets
   a. Common Crawl
   b. Wikipedia
4. Pipeline
   a. Presentation
   b. Algorithms
5. Results

Problem presentation

The Common Crawl is an **open repository of web crawl data** collected over last 7 years.

The goal is to create a linguistic repository by **categorizing the common crawl data to appropriate languages**, focusing on **rare languages**

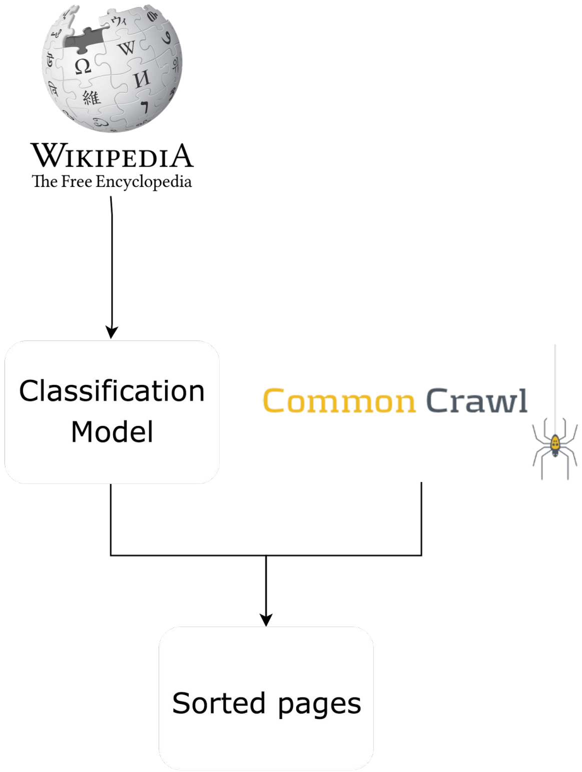
Images credit: Common Crawl project

Architecture

Train the language classifier on **Wikipedia data**.

Process archive files **in parallel**: unpack, process, delete unpacked version.

When a page in a rare language (i.e. not in the top 10) is found, **save** its URL, plaintext and detected language on mongoDB.



Architecture: Technologies

Common Crawl data: stored on **Amazon S3**. We use **Globus** for data transfers.

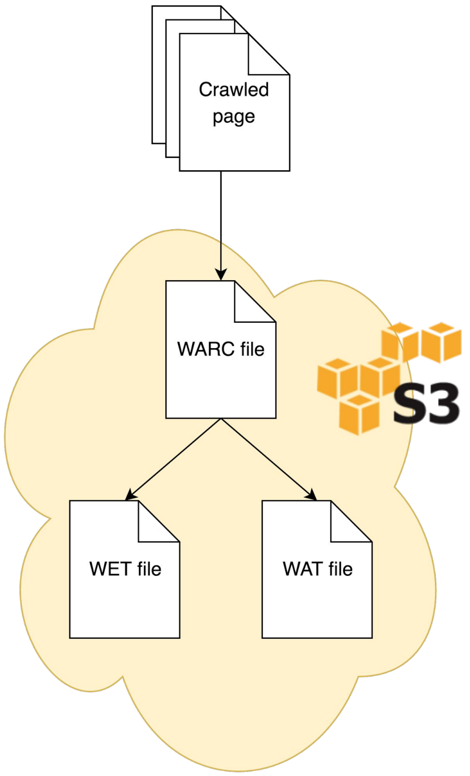
ML pipeline: **PySpark**.

Storage: **MongoDB** ?

Datasets: Common Crawl file formats

Stored as gzipped archive files, with many pages in each file

3 main file formats: **WARC**, **WAT** and **WET**.



Datasets: Common Crawl file formats

WET contains the extracted page text, could be useful for classification.

Extraction algorithm is available online, but probably hard to implement in a Spark pipeline.

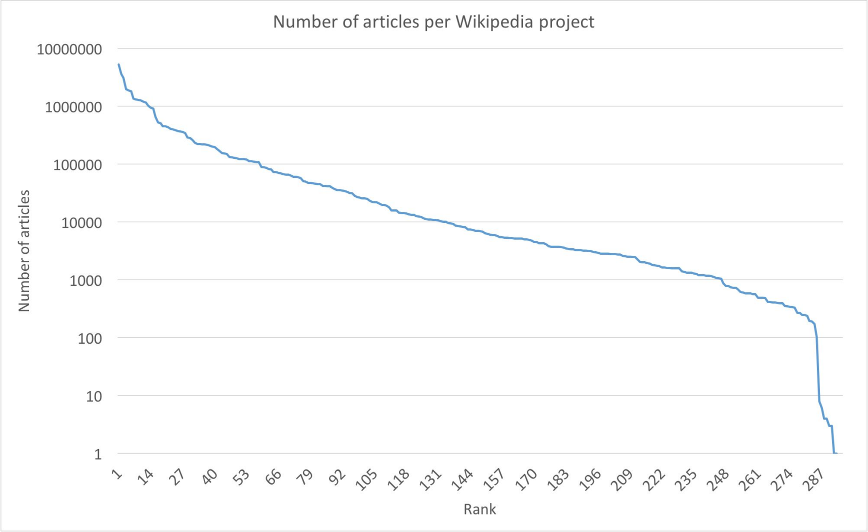
Other possibility: strip HTML ourselves.

Datasets: Wikipedia

295 languages, total of 42,000,000 articles.

Sizes vary:

- 49 Wikipedias under 1,000 articles
- 112 between 1,000 and 10,000 articles



Datasets: Wikipedia

Quality also varies

Some editions are mostly **bot-generated** articles, e.g.

- Plant and animal articles from templates+standardized databases
- Machine-translated pages

Some editions have many articles, but mostly **very short stubs.**

Typical bot-generated page on the Cebuano Wikipedia

Datasets: Wikipedia

Available as regularly-updated database dumps.

Total size of the compressed dumps for all 295 languages is 53GB, or about 264GB of uncompressed XML data.

The pages are in Wikicode (a markup language), we use an external script to strip the markup and turn the dumps into (almost) plain text files.

Pipeline

Data retrieval and preprocessing using custom Bash+Python scripts

Actual classification implemented in pySpark

Basic pipeline:

```
Tokenizer -> CountVectorizer -> NaiveBayes
```

Pipeline: Algorithms

Naive Bayes on unigrams gives good performance

Probably due to the fact that different languages have few words in common

Requires a lot of memory because there are a lot of different words across all languages.

Contrary to normal document classification, Heap’s law doesn’t apply.

No IDF term used

Pipeline: Algorithms

- Naive Bayes on bigrams
  - Significantly longer, uses much more memory
  - Not a very significant performance improvement
  - Also probably due to the fact that most of the information is already available from the unigrams.
- Decision Tree
  - We weren’t able to train one for lack of RAM

Pipeline: Algorithms

Naive Bayes on letter trigrams

- Avoids the problem of having an enormous vocabulary
- Fast training with reasonably small memory use
- Terrible results

Results

Confusion Matrix: Diagonal elements are correctly predicted languages (true positives)
Overall Accuracy: 90.84%

v lang / pr	simplewiki	htwiki	bpywiki	iawiki	bat_smgw	wawiki	napwiki	gdwiki	bugwiki	amwiki	map_bmsv	mznwiki	nahwiki	azbwiki	hsbwiki	mrjwiki	oswiki	bowiki	hifwiki	mhrwiki
simplewiki	23743	224	0	2	30	6	2	0	0	0	6	0	21	0	0	0	0	70	10	0
htwiki	10	10362	0	3	2	1	2	1	0	0	0	0	5	0	0	0	0	0	3	0
bpywiki	0	4	5109	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
iawiki	24	489	0	3391	24	5	4	2	1	1	0	0	12	0	1	1	0	8	0	0
bat_smgw	0	3	0	0	3163	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
wawiki	13	95	0	2	2	2785	1	1	0	0	0	0	0	0	0	0	0	3	1	0
napwiki	43	330	0	3	38	2	2490	0	0	0	0	0	9	1	0	1	0	0	0	0
gdwiki	5	233	0	0	52	2	1	2435	0	0	0	0	17	0	1	6	0	1	2	3
bugwiki	0	2	0	0	1	0	0	0	2734	0	6	0	0	0	0	0	0	0	0	0
amwiki	6	47	0	0	1	0	1	1	0	2683	0	0	0	0	0	0	0	0	0	0
map_bmsv	45	313	0	0	18	6	2	0	0	0	2291	0	6	0	1	0	0	2	0	0
mznwiki	1	24	0	0	0	0	0	0	0	0	0	2476	0	4	0	0	0	0	0	0
nahwiki	9	38	0	1	2	1	0	0	0	0	1	0	2168	0	0	0	0	4	0	0
azbwiki	1	13	0	0	0	0	0	0	0	0	0	27	0	2018	0	0	0	2	1	0
hsbwiki	35	26	0	0	2	1	1	0	0	0	0	0	0	0	2163	0	0	0	0	0
mrjwiki	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	2047	0	0	0	5
oswiki	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1871	0	0	2
bowiki	31	1131	0	0	19	7	4	0	0	0	0	0	6	0	0	2	0	815	1	1
hifwiki	20	179	3	0	5	0	0	1	1	0	3	0	2	0	0	0	0	4	1728	0
mhrwiki	44	69	0	0	5	2	1	0	0	0	0	0	0	0	0	3	1	2	0	1784

Results

F1 Measure: In case of binary classification, it is the equally weighted harmonic mean of Precision and Recall.

In case of multiclass classification problems, assuming that we have a one vs all classifier, we can compute 2 types of metrics: Micro-averaged and Macro-averaged.

Macro-averaged F1 score is preferred to understand the classifier's performance for rare languages as it gives equal weights to all the classes performance.

Here precision (macro) = 0.970, recall (macro) = 0.835 (calculated as average of the precision and recall of all the classes)

F1- Score (Macro averaged) = 0.897 (Ranges from 0-1)