Corpus Acquisition from the Internet

Philipp Koehn
partially based on slides from Christian Buck

12 November 2020
For many language pairs, lots of text available.

Text you read in your lifetime: 300 million words

Translated text available: billions of words

English text available: trillions of words
Mine the Web

• Largest source for text: the World Wide Web

Common Crawl

– publicly available crawl of the web
– hosted by Amazon Web Services, but can be downloaded
– regularly updated (semi-annual)
– 2-4 billion web pages per crawl

• Currently filling up hard drives in our lab
Monolingual Data

- Starting point: 35TB of text

- Processing pipeline [Buck et al., 2014]
  - language detection
  - deduplication
  - normalization of Unicode characters
  - sentence splitting

- Obtained corpora

<table>
<thead>
<tr>
<th>Language</th>
<th>Lines (B)</th>
<th>Tokens (B)</th>
<th>Bytes</th>
<th>BLEU (WMT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>59.13</td>
<td>975.63</td>
<td>5.14 TB</td>
<td>-</td>
</tr>
<tr>
<td>German</td>
<td>3.87</td>
<td>51.93</td>
<td>317.46 GB</td>
<td>+0.5</td>
</tr>
<tr>
<td>Spanish</td>
<td>3.50</td>
<td>62.21</td>
<td>337.16 GB</td>
<td>-</td>
</tr>
<tr>
<td>French</td>
<td>3.04</td>
<td>49.31</td>
<td>273.96 GB</td>
<td>+0.6</td>
</tr>
<tr>
<td>Russian</td>
<td>1.79</td>
<td>21.41</td>
<td>220.62 GB</td>
<td>+1.2</td>
</tr>
<tr>
<td>Czech</td>
<td>0.47</td>
<td>5.79</td>
<td>34.67 GB</td>
<td>+0.6</td>
</tr>
</tbody>
</table>
Parallel Data

• Basic processing pipeline [Smith et al., 2013]
  – find parallel web pages (based on URL only)
  – align document by HTML structure
  – sentence splitting and tokenization
  – sentence alignment
  – filtering (remove boilerplate)

• Obtained corpora

<table>
<thead>
<tr>
<th>Language</th>
<th>French</th>
<th>German</th>
<th>Spanish</th>
<th>Russian</th>
<th>Japanese</th>
<th>Chinese</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments</td>
<td>10.2M</td>
<td>7.50M</td>
<td>5.67M</td>
<td>3.58M</td>
<td>1.70M</td>
<td>1.42M</td>
</tr>
<tr>
<td>Foreign Tokens</td>
<td>128M</td>
<td>79.9M</td>
<td>71.5M</td>
<td>34.7M</td>
<td>9.91M</td>
<td>8.14M</td>
</tr>
<tr>
<td>English Tokens</td>
<td>118M</td>
<td>87.5M</td>
<td>67.6M</td>
<td>36.7M</td>
<td>19.1M</td>
<td>14.8M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Bengali</th>
<th>Farsi</th>
<th>Telugu</th>
<th>Somali</th>
<th>Kannada</th>
<th>Pashto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments</td>
<td>59.9K</td>
<td>44.2K</td>
<td>50.6K</td>
<td>52.6K</td>
<td>34.5K</td>
<td>28.0K</td>
</tr>
<tr>
<td>Foreign Tokens</td>
<td>573K</td>
<td>477K</td>
<td>336K</td>
<td>318K</td>
<td>305K</td>
<td>208K</td>
</tr>
<tr>
<td>English Tokens</td>
<td>537K</td>
<td>459K</td>
<td>358K</td>
<td>325K</td>
<td>297K</td>
<td>218K</td>
</tr>
</tbody>
</table>

• Much more work needed!
Data Cleaning and Subsampling

• Not all data useful – some may be harmful

• Removing data based on
  – domain relevance
  – alignment quality
  – redundancy
  – bad language (orthography, non-words)
  – machine translated or poorly translated

• Removing bad data always reduces training time

• Removing bad data sometimes helps quality

• Clean data approach (only using high quality data) helps in limited domains
corpus crawling
Finding Monolingual Text

• Simple Idea
  1. Download many websites
  2. Extract text from HTML
  3. Guess language of text
  4. Add to corpus
  5. Profit

• Turns out all these steps are quite involved
Common Crawl

- Non-profit organization

- Data
  - publicly available on Amazon S3
  - e.g. January 2015: 140TB / 1.8B pages

- Crawler
  - Apache Nutch
  - collecting pre-defined list of URLs
extracting text
Bash Shell: Find Out Linux / FreeBSD / UNIX System Load Average

by nixcraft on March 23, 2005 - 8 Comments - Last Updated August 8, 2013
in Linux, Monitoring, Sys Admin

Yes, I know we can use the `uptime` command to find out the system load average. The `uptime` command displays the current time, the length of time the system has been up, the number of users, and the load average of the system over the last 1, 5, and 15 minutes. However, if you try to use the `uptime` command in script, you know how difficult it is to get correct load average. As the time since the last, reboot moves from minutes, to hours, and an even day after system rebooted. Just type the `uptime` command:

```
$ uptime
```
Y es, I know we can use the `<kbd>uptime</kbd>` command to find out the system load average. The `uptime` command displays the current time, the length of time the system has been up, the number of users, and the load average of the system over the last 1, 5, and 15 minutes. However, if you try to use the `uptime` command in script, you know how difficult it is to get correct load average. As the time since the last, reboot moves from minutes, to hours, and an even day after system rebooted. Just type the `uptime` command:

```bash
$ uptime
```

Sample outputs:

```
0:09:01 up 29 min, 1 user, load average: 0.00, 0.00, 0.00
```

Traditionally, UNIX administrators used `sed` and other shell command in scripting to get correct value of load average. Here is my own modified hack to save the time:

```bash
$ uptime | awk -F'load averages:' '{ print $2 }
```

OR better use the following code:

```bash
$ uptime | awk -F'[a-z:]' '{ print $2 }
```

Output taken from my `<strong>OS X desktop</strong>`:

```
1.24 1.34 1.35
```

Output taken from my `<strong>Ubuntu</strong>` Linux server:

```
0.00, 0.01, 0.05
```

Output taken from my `<strong>RHEL</strong>` based server:

```
0.24, 0.27, 0.21
```

Output taken from my `<strong>FreeBSD</strong>` based server:

```
0.71, 0.71, 0.58
```

Please note that command works on all variant of UNIX operating systems. See also:

```bash
```

`chksysload.bash`
LAST UPDATED August 8, 2013 in Linux , Monitoring , Sys admin

Yes, I know we can use the uptime command to find out the system load average. The uptime command displays the current time, the length of time the system has been up, the number of users, and the load average of the system over the last 1, 5, and 15 minutes. However, if you try to use the uptime command in script, you know how difficult it is to get correct load average. As the time since the last reboot moves from minutes, to hours, and an even day after system rebooted. Just type the uptime command: $ uptime Sample outputs: 1:09:01 up 29 min, 1 user, load average: 0.00, 0.00, 0.00
LAST UPDATED August 8, 2013

Yes, I know we can use the `uptime` command to find out the system load average. The `uptime` command displays the current time, the length of time the system has been up, the number of users, and the load average of the system over the last 1, 5, and 15 minutes. However, if you try to use the `uptime` command in script, you know how difficult it is to get correct load average. As the time since the last, reboot moves from minutes, to hours, and an even day after system rebooted. Just type the `uptime` command:

```
$ uptime
```

Sample outputs: `1:09:01 up 29 min, 1 user, load average: 0.00, 0.00, 0.00`
language detection
Muitas intervenções alertaram para o facto de a política dos sucessivos governos PS, PSD e CDS, com cortes no financiamento das instituições do Ensino Superior e com a progressiva desresponsabilização do Estado das suas funções, ter conduzido a uma realidade de destruição da qualidade do Ensino Superior público.
Muitas intervenções alertaram para o facto de a política dos sucessivos governos PS, PSD e CDS, com cortes no financiamento das instituições do Ensino Superior e com a progressiva desresponsabilização do Estado das suas funções, ter conduzido a uma realidade de destruição da qualidade do Ensino Superior público.
Example: langid.py

- Muitas intervenções alertaram
  - prediction: Portuguese
  - high confidence (-90.8)

- Muitas intervenções
  - prediction: Portuguese
  - fairly high confidence (-68.2)

- Muitas
  - prediction: English
  - low confidence (9.1)
Language Identification Tools

• **langid.py** (Lui & Baldwin, ACL 2012)
  – 1-4 grams, NaiveBayes, Feature Selection

• **TextCat** (based on Cavnar & Trenkle, 1994)
  – similar to langid.py
  – no Feature Selection

• **Compact/Chromium Language Detector 2** (Google)
  – takes hints from tld, meta data
  – super fast
  – detects spans of text
Detected Languages in CommonCrawl

(Buck and Heafield, LREC2014)
## Most Common English Phrases

<table>
<thead>
<tr>
<th>Count (M)</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1374.44</td>
<td>Add to</td>
</tr>
<tr>
<td>816.33</td>
<td>Share</td>
</tr>
<tr>
<td>711.68</td>
<td>Unblock User</td>
</tr>
<tr>
<td>68.31</td>
<td>Sign in or sign up now!</td>
</tr>
<tr>
<td>61.26</td>
<td>Log in</td>
</tr>
<tr>
<td>54.77</td>
<td>Privacy Policy</td>
</tr>
<tr>
<td>45.18</td>
<td>April 2010</td>
</tr>
<tr>
<td>34.35</td>
<td>Load more suggestions</td>
</tr>
<tr>
<td>19.84</td>
<td>Buy It Now</td>
</tr>
<tr>
<td>16.64</td>
<td>Powered by WordPress.com</td>
</tr>
</tbody>
</table>
## Benefit of Huge Language Models

<table>
<thead>
<tr>
<th></th>
<th>2012</th>
<th>Δ</th>
<th>2013</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>35.8</td>
<td></td>
<td>30.9</td>
<td></td>
</tr>
<tr>
<td>+ 50M lines</td>
<td>36.3</td>
<td>0.5</td>
<td>31.5</td>
<td>0.6</td>
</tr>
<tr>
<td>+ 100M lines</td>
<td>36.5</td>
<td>0.7</td>
<td>31.5</td>
<td>0.6</td>
</tr>
<tr>
<td>+ 200M lines</td>
<td>36.6</td>
<td>0.8</td>
<td>31.8</td>
<td>0.9</td>
</tr>
<tr>
<td>+ 400M lines</td>
<td>37.0</td>
<td>1.2</td>
<td>31.8</td>
<td>0.9</td>
</tr>
<tr>
<td>+ 800M lines</td>
<td>37.3</td>
<td>1.6</td>
<td>31.8</td>
<td>0.9</td>
</tr>
<tr>
<td>+ 1.3B lines</td>
<td>37.7</td>
<td>1.9</td>
<td>32.0</td>
<td>1.1</td>
</tr>
</tbody>
</table>
bilingual corpus crawling
• Bilingual text = same text in different languages
• Usually: one side translation of the other
• Full page or interface/content only
• Potentially translation on same page
  e.g., Twitter, Facebook posts
Pipeline

1. Identify web sites worth crawling
2. Crawl web site
3. Language detection — as before
4. Extract text from HTML — as before
5. Align documents
6. Align sentences
7. Clean corpus
identify web sites
Targeted Crawling

- A few web sites with a lot of parallel text, e.g.,
  - European Union, e.g., proceedings of the European Parliament
  - Canadian Hansards
  - United Nations
  - Project Syndicate
  - TED Talks
  - Movie / TV show subtitles
  - Global Voices

- Hand-written tools
  - crawling
  - text extraction
  - document alignment

- Few days effort per site
Broad Crawling

- Identify many web sites to crawl
  - has the phrase This page in English or variants
  - has link to language flag
  - known to have content in multiple languages (from CommonCrawl)

- Follow links
  - up to $n$ links deep into site
  - up to $n$ links in total
  - only follow links to web pages, not images, etc.

- Avoid crawling sites too deeply that do not have parallel text?
  (requires quick feedback from downstream processing)
document alignment
Document Alignment

• Early Work: STRAND (Resnik 1998, 1999)
  (Structural Translation Recognition, Acquiring Natural Data)

• Pipeline

  1. candidate generation
  2. candidate ranking
  3. filtering
  4. optional: sentence alignment
  5. evaluation
Link Structure

- Parent page: a page that links to different language versions

```
English French Spanish
x.com/en/cat.html
x.com/fr/chat.html
```
Parent Page Example
• A page that links to its translation in another language
- Often URLs differ only slightly, often indicating language

<table>
<thead>
<tr>
<th>xyz.com/en/</th>
<th>xyz.com/fr/</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyz.com/bla.htm</td>
<td>xyz.com/bla.htm?lang=FR</td>
</tr>
<tr>
<td>xyz.com/the_cat</td>
<td>xyz.fr/le_chat</td>
</tr>
</tbody>
</table>
### Finding URL Patterns

- **URLs with pattern =en**

<table>
<thead>
<tr>
<th>Count</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>545875</td>
<td>lang=en</td>
</tr>
<tr>
<td>140420</td>
<td>lng=en</td>
</tr>
<tr>
<td>126434</td>
<td>LANG=en</td>
</tr>
<tr>
<td>110639</td>
<td>hl=en</td>
</tr>
<tr>
<td>99065</td>
<td>language=en</td>
</tr>
<tr>
<td>81471</td>
<td>t1ng=en</td>
</tr>
<tr>
<td>56968</td>
<td>l=en</td>
</tr>
<tr>
<td>47504</td>
<td>locale=en</td>
</tr>
<tr>
<td>33656</td>
<td>langue=en</td>
</tr>
<tr>
<td>33503</td>
<td>lang=eng</td>
</tr>
<tr>
<td>19421</td>
<td>uil=English</td>
</tr>
<tr>
<td>15170</td>
<td>ln=en</td>
</tr>
<tr>
<td>14242</td>
<td>Language=EN</td>
</tr>
<tr>
<td>13948</td>
<td>lang=EN</td>
</tr>
<tr>
<td>12108</td>
<td>language=english</td>
</tr>
<tr>
<td>11997</td>
<td>lang=engcro</td>
</tr>
<tr>
<td>11646</td>
<td>store=en</td>
</tr>
</tbody>
</table>
Finding URL Patterns

- URLs with pattern `lang.*=.*`

<table>
<thead>
<tr>
<th>Count</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>13948</td>
<td>lang=EN</td>
</tr>
<tr>
<td>13456</td>
<td>language=ca</td>
</tr>
<tr>
<td>13098</td>
<td>switchlang=1</td>
</tr>
<tr>
<td>12960</td>
<td>language=zh</td>
</tr>
<tr>
<td>12890</td>
<td>lang=Spanish</td>
</tr>
<tr>
<td>12471</td>
<td>lang=th</td>
</tr>
<tr>
<td>12266</td>
<td>langBox=US</td>
</tr>
<tr>
<td>12108</td>
<td>language=english</td>
</tr>
<tr>
<td>12003</td>
<td>lang=cz</td>
</tr>
<tr>
<td>11997</td>
<td>lang=engcro</td>
</tr>
<tr>
<td>11635</td>
<td>lang=sl</td>
</tr>
<tr>
<td>11578</td>
<td>lang=d</td>
</tr>
<tr>
<td>11474</td>
<td>lang=lv</td>
</tr>
<tr>
<td>11376</td>
<td>lang=NL</td>
</tr>
<tr>
<td>11349</td>
<td>lang=croeng</td>
</tr>
<tr>
<td>11244</td>
<td>lang=English</td>
</tr>
</tbody>
</table>
Document Length

- Extract texts and compare lengths (Smith 2001)

\[
\text{Length}(E) \approx C \times \text{Length}(F)
\]

learned,
language-specific parameter

- Document or sentence level
Document Object Model

- Translated web pages often retain similar structure

```html
<html>
<body>
<h1>Where is the cat?</h1>
The cat sat on the mat.
</body>
</html>

<html>
<body>
El gato se sentó en la alfombra.
</body>
</html>
```

- This includes links to the same images, etc.
Linearized Structure

[Start:html]
[Start:body]
[Start:h1]
[Chunk:17bytes]
[End:h1]
[Chunk:23bytes]
[End:body]
[End:html]
### Levenshtein Alignment

<table>
<thead>
<tr>
<th>Segment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:html</td>
<td>Keep</td>
</tr>
<tr>
<td>Start:body</td>
<td>Keep</td>
</tr>
<tr>
<td>Start:h1</td>
<td>Delete</td>
</tr>
<tr>
<td>Chunk:17bytes</td>
<td>Delete</td>
</tr>
<tr>
<td>End:h1</td>
<td>Delete</td>
</tr>
<tr>
<td>Chunk:23bytes</td>
<td><strong>23 Bytes -&gt; 32 Bytes</strong></td>
</tr>
<tr>
<td>End:body</td>
<td>Keep</td>
</tr>
<tr>
<td>End:html</td>
<td>Keep</td>
</tr>
</tbody>
</table>
Content Similarity

• Simple things
  – same numbers or names in documents
  – often quite effective

• Use of lexicon
  – treat documents as bag of words
  – consider how many words in EN document have translations in FR document

• A bit more complex
  – semantic representations of documents content
  – bag of word vectors
  – neural network embeddings

• Major challenge: do this fast for $n \times m$ document pairs
Google’s Content Matching

• Basic idea: translate everything into English, match large n-grams

• For each non-English document:
  1. Translate everything to English using MT
  2. Find distinctive ngrams
     (a) rare, but not too rare (5-grams)
     (b) used for matching only

• Build inverted index: ngram → documents

  [cat sat on] → {[doc₁, ES], [doc₃, DE], ...}
  [on the mat] → {[doc₁, ES], [doc₂, FR], ...}
Matching using Inverted Index

\[
\begin{align*}
\text{[cat sat on]} & \rightarrow \{[\text{doc}_1, \text{ES}], [\text{doc}_3, \text{DE}], \ldots\} \\
\text{[on the mat]} & \rightarrow \{[\text{doc}_1, \text{ES}], [\text{doc}_2, \text{ES}], \ldots\} \\
\text{[on the table]} & \rightarrow \{[\text{doc}_3, \text{DE}]\}
\end{align*}
\]

- For each n-gram
  - generate all pairs where:
    * document list short (≤ 50)
    * source language different

- Result: \([\text{doc}_1, \text{doc}_3], \ldots\)
Scoring using Forward Index

- Forward index maps documents to n-grams
- For each document pair $[d_1, d_2]$
  - collect scoring n-grams for both documents
  - build IDF-weighted vector
  - distance: cosine similarity
Scoring Document Pairs

• Given
  \[\text{ngrams}(d_1) = n_1, n_2, \ldots, n_r\]
  \[\text{ngrams}(d_2) = n'_1, n'_2, \ldots, n'_r,\]

• Inverse document frequency
  \[\text{idf}(n) = \log \frac{|D|}{df(n)}\]
  where: \(|D| = \text{number of documents}\]
  \[df(n) = \text{number of documents with } n\]

• Scoring of IDF-weighted vectors \(v\)
  \[v_{1,x} = \text{idf}(n_x) \text{ if } n_x \in \text{ngrams}(d_1), 0 \text{ otherwise}\]
  \[v_{2,x} = \text{idf}(n_x) \text{ if } n_x \in \text{ngrams}(d_2), 0 \text{ otherwise}\]
  \[\text{score}(d_1, d_2) = \frac{v_1 \cdot v_2}{||v_1|| ||v_2||}\]
sentence alignment
Sentence Alignment

- Much early work in 1990s, e.g., Gale and Church (1991)
  - find sequence of 1-1, 1-2, 0-1, etc., sentence alignment groups
  - good element in sequence = similar number of words
  - dynamic programming search for best sequence

- Featurized alignments
  - with dictionary (Hunalign)
  - with induced dictionary (Gargantua)
  - consider tags such as $<$P$>$

- Sensitive to noise — often large parts of page not translated
Sentence Pair Similarity

- Core Problem: both sentences must have same meaning
- Translate foreign sentence into English
  measure similarity with metrics like BLEU
- Words in one sentence have translation in the other
- Cross-lingual sentence embeddings
Sentence Embeddings

- **LASER**: Neural machine translation model with bottleneck feature
Sentence Embeddings

The dog is brown.
Le chien est brun.

I love eating.
Ich esse gerne.

I enjoy food a lot.
Ich genieße Essen.

I want to call you.
Ich will dich anrufen.
Vecalign

- Uses LASER sentence embeddings
- Linear time coarse-to-fine algorithm
sentence pair filtering
Filtering Bad Data

- Mismatched sentence pairs from errors in pipeline

- Non-literal translation
e.g. news stories are notoriously non-literal

- Bad translations

- Machine translation
  - much of the parallel text on the Internet generated by Google Translate
  - detection hard — looks like very clean parallel data
  - maybe too clean (little reordering, very literal)
  - watermarking machine translation (Venugopal et al., 2011)

- How clean should it be?
  - trade-off between precision and recall unclear
Methods

• Dual cross-entropy
  – view sentence pair as input/output
  – score with neural machine translation model in both directions
  – scores should be slow and similar

• LASER embeddings

• Feature-based approaches
  – matching numbers, named entities
  – language model probabilities
  – lexical translation probabilities

• Classifier
  – positive example: sentence pair from clean corpus
  – negative example: corrupted example (misalignment, words changed, ...)
Open Challenges

• Currently serious attempt at broad crawling for parallel data at JHU

• Major challenges
  – crawling (just using standard tool)
  – document alignment (major research topic)
    → shared task at WMT 2016 machine translation conference
  – sentence alignment (just using standard tool)
  – detection of machine translated text (some old work)
  – filtering out bad sentence pairs (major research topic)
    → shared tasks at WMT 2018–2020 machine translation conference

• JHU efforts (Paracrawl): continuously processing terabytes of data