COR: Corpus Linquistics

Lecture 6
DIY Corpora, Processing Raw Text, SQL

Francis Bond
Department of Asian Studies
Palacký University
https://fcbond.github.io/
bond@ieee.org

https://github.com/bond-lab/Corpus-Linguistics

Overview

- DIY Corpora
- Processing Raw Text
- Structured Query Language

DIY Corpora

Why build your own?

1. Decide what you want to study
   see if you can do it with existing resources
   No ☹

2. Collect data that fits your needs

   - ➢ Speech (expensive)
   - ➢ Text (easy to get, hard to do legally)

3. Process it

   - ➢ Clean up
   - ➢ Mark up (what do we know about the data originally)
   - ➢ Annotation (what will we add to it)

Collecting Text

Some ways to collect text:

- Word-processed texts: save as a text only file
- Keyboard entry: speech transcription, students' handwritten essays, etc.
- Scanning: copyright protected novels, latest magazines, etc.
- CD-ROM: newspaper, encyclopaedia, ICAME, etc.
- Internet resources: email, chat, public documents, newspapers, magazines etc.
- Text archives: copyright free (old) novels, essays, etc.
- Copying from a large corpus: e.g. using sections of the BNC

Recent Trends

- Government Documents (proceedings, publications)
- Open source documents (manuals, wikipedia)
- Social Media: Blogs and twitter

Web as Corpus

Two Approaches to using the Web as a Corpus

- Direct Query: Search Engine as Query tool and WWW as corpus?
  (Objection: Results are not reliable)
  - Population and exact hit counts are unknown → no statistics possible.
  - Indexing does not allow to draw conclusions on the data.
  - ⊗ Google is missing functionalities that linguists / lexicographers would like to have.

- Web Sample: Use search engine to download data from the net and build a corpus from it.
  - known size and exact hit counts → statistics possible.
  - people can draw conclusions over the included text types.
  - (limited) control over the content.
  - ⊗ sparser data

Direct Query

- Accessible through search engines (Google API, Yahoo API, Scripts)
- Document counts are shown to correlate directly with "real" frequencies (Keller 2003), so search engines can help - but...
  - lots of repetitions of the same text (not representative)
  - very limited query precision (no upper/lower case, no punctuation...)
  - only estimated counts, often hard to reproduce exactly
  - different queries give wildly different numbers

Web Sample

- Extracting and filtering web documents to create linguistically annotated corpora (Kilgarriff 2006)
  - gather documents for different topics (balance!)
  - exclude documents which cannot be preprocessed with available tools (here taggers and lemmatizers)
  - exclude documents which seem irrelevant for a corpus (too short or too long, word lists,...)
  - do this for several languages and make the corpora available

Building Internet Corpora: Outline

1. Select Seed Words (500)

2. Combine to form multiple queries (6,000)

3. Query a search engine and retrieve the URLs (50,000)

4. Download the files from the URLS (100,000,000 words)

5. Postprocess the data (encoding; cleanup; tagging and parsing)

Sharoff, S (2006) Creating general-purpose corpora using automated search engine queries. In M. Baroni, S. Bernardini (eds.) WaCky! Working papers on the Web as Corpus, Bologna, 2006.

Post-processing

- Filter documents by size
  - Small documents ($< 5KB$) contain very little real text
  - Large documents ($> 200KB$) tend to be indices, catalogues, lists, etc.
- Remove perfect duplicates
  - Actually, removed both the original & the duplicate:

  ... tend to be warning messages

Boilerplate stripping

- Boilerplate: HTML markup, javascript, other non-linguistic material
- Removing boilerplate information is crucial to obtaining linguistic data only
  - Content-rich sections of a document will have a low html tag density
  - Boilerplate sections have a wealth of html
  - This heuristic is "relatively independent of language and crawling strategy"
- If a text does not have enough function words, it is likely non-linguistic material (e.g., a list)
  - Require at least 10 function word types and 30 tokens on a page
  - ... which must make up at least 25% of the total words

Near-duplicate detection

- Take fingerprints' of a fixed number of randomly-selected n-grams (ignoring function words)
  - e.g., extract 25 5-grams from each document
- Near-duplicates have a high overlap
  - e.g., at least 2 5-grams in common

Linguistic Post-processing

- Prepare the data for searching:
  - Run a POS tagger over it
  - Clean the documents further, using POS tags
    - Where the POS tag distribution is unusual,
    - ... perform another round of anomalous document finding
    - Look for problematic (erroneous) POS tags and remove those documents
    - Use cues such as number of unrecognized words, proporition of words with upper-case initial letters, ...
- Index the document by word, POS and lemma

Internet Corpora Summary

- ➢ The web can be used as a corpus
  - ➢ Direct access
    - ∗ Fast and convenient
    - ∗ Huge amounts of data
    - ⊗ unreliable counts
  - ➢ Web sample
    - ∗ Control over the sample
    - ∗ Some setup costs (semi-automated)
    - ⊗ Less data
- ➢ Richer data than a compiled corpus
- ⊗ Less balanced, less markup

Processing Raw Text

Language Identification

- Given a document and a list of possible languages, in what language was the document written? (e.g. English, German, Japanese, Uyghur, ...)
- Language identification provides us with the means to automatically "discover" web data to convert into a corpus over which to learn linguistic (lexical) properties
- Main Approaches
  - Linguistically-grounded methods
    - Diacritics/Characters
    - Character n-grams
    - Stop words
  - Statistical Methods
  - Context (under `.jp` or `.ko`?)

Normalization

- Extracting text from various documents
- Segmenting continuous text
- Number Normalization: *$700K*, *$700,000*, *0.7 million dollars*, ...
- Date Normalization: *2000AD, 1421AH, Heisei 12, ...*
- Stripping stop words
- Lemmatization: *produces* $\rightarrow$ *produce*
- Stemming: *producer* $\rightarrow$ *produc*; *produces* $\rightarrow$ *produc*
- Decompounding: *zonnecel* $\rightarrow$ *zon cel*

Relational Databases and SQL

Many corpora are stored as Relational Data-Bases

- Data is stored in **Table**s (or relations)
  - **attribute**s are **column**s
  - **record**s are **row**s
- Tables can be joined together
  - If they share a common column
- Data can be retrieved with **Queries**
  - Properly written these can be very fast

Structured Query Language (SQL) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS)

Example tables

TABLE: sent

sid	sent
INTEGER	STRING
1	Does this work?
2	I hope so.

TABLE: word

sid	wid	word	pos	lemma
INTEGER	INTEGER	STRING	STRING	STRING
1	1	Does	VBZ	do
1	2	this	DT	this
1	3	work	NN	work
1	4	?	.	?
2	1	I	PRN	i
...				

Select-From-Where

- The simplest query
  - SELECT desired attributes (columns)
  - FROM one or more tables
  - WHERE condition applies about the records in the tables
- What lemmas are there associated with the word *does*?

```
SELECT word, lemma
FROM word
WHERE word = 'does'
```

word	lemma
does	do
does	do
does	doe
...	

How does it work?

- Begin with the table in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Select only those parts indicated by the SELECT clause.

Note: each query ends with a semicolon ";", not shown in the example

* in SELECT

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."

- What information is there associated with the word *does*?

```
SELECT *
FROM word
WHERE word = 'does'
```

sid	wid	word	pos	lemma ...
10	1	does	VBZ	do
11	7	does	VBZ	do
14	4	does	NNS	doe
...				

You can rename things

- What information is there associated with the word *does*?

```
SELECT word AS surface, lemma AS indexform, pos
FROM word
WHERE word = 'does'
```

surface	indexform	pos
does	do	VBZ
does	do	VBZ
does	doe	NNS
...		

You can sort things

- What information is there associated with the word *does*, sorted by POS?

```
SELECT word AS surface, lemma AS indexform, pos
FROM word
WHERE word = 'does'
ORDER BY pos DESC
```

surface	indexform	pos
does	do	VBZ
does	do	VBZ
does	doe	NNS
...		

- You can specify the order with `DESC` or `ASC` (default)
- You can order by multiple things: `ORDER BY pos, LENGTH(word)`

And add new things

- What is the lemma, its length and pos associated with the word *does*?

```
SELECT  lemma, LENGTH(lemma) AS length, pos
FROM word
WHERE word = 'does'
```

lemma	length	pos
do	2	VBZ
do	2	VBZ
doe	3	NNS
...		

- strings: `LENGTH()`, `LOWER()`, `UPPER()`, `TRIM()`, `LTRIM()`, `RTRIM()` `SUBSTR(str,from, len)`, `REPLACE(str, from, into)`
- numbers: `ROUND(num,digits=0)`, `ABS(num)`

Task

- Download `eng.db` and `cmn.db`
- Start the sqlitebrowser
- Open *eng.db*
  - Find all surface forms of ***leave***
  - Find all surface forms of ***leave***, and show their length
  - Find all surface forms of ***leave***, ordered from longest to shortest

1. Open the database
2. Look at the database structure
3. Execute SQL

You can have complex conditions and limits

- Show 5 words that are not the same as their lemmas:

```
SELECT  word, lemma, pos
FROM word
WHERE word != lemma
LIMIT 5
```

word	lemma	pos
does	do	VBZ
held	hold	VBN
does	does	NNS
Holmes	holmes	NNP
Holmes	holmes	NNP
...		

You can even have simple regular expressions

- Show 5 words that include the string dog:

```
SELECT  word, lemma, pos
FROM word
WHERE word GLOB '*dog*'
LIMIT 5
```

word	lemma	pos
dog-cart	dog-cart	NN
dog	dog	NN
dog-	dog	NN
dog-cart	dog-cart	NN
dogged	dog	VBD
...		

Or slightly complicated ones

- Show 5 words starting with dog or Dog whose part of speech is not a noun

```
SELECT  word, lemma, pos
FROM word
WHERE word GLOB '[dD]og*' AND pos NOT GLOB 'N*'
LIMIT 5
```

word	lemma	pos
dogged	dog	VBN

* matches anything
? matches one or one of anything
[ ] matches all characters listed (and allows ^ for negation and - for ranges)
e.g., [a-z], [^H],

You can aggregate results

- Tell me more about the words

```
SELECT COUNT(word), COUNT(DISTINCT word),
       MIN(LENGTH(word)),
       MAX(LENGTH(word)), AVG(LENGTH(word))
FROM word
```

count	distinct	min	max	avg
77820	9485	1	86	4.437

- `MIN(num)`, `AVG(num)`, `MAX(num)` are calculated over the whole set

- `DISTINCT` eliminates duplicate records thus fetches only unique records

You can group things

- Tell me more about the words grouped into parts-of-speech

```
SELECT pos, word,
       count(word), count(DISTINCT word),
       MIN(LENGTH(word)),
       MAX(LENGTH(word)), AVG(LENGTH(word))
FROM word
GROUP BY pos
```

pos	word	count	distinct	min	max	avg
CC	and	2264	13	2	7	2.98
DT	the	8256	38	1	7	2.74
EX	There	243	2	5	5	5.00
...						

Task

- Which POS has the greatest difference between lemma and word length?
- Which preposition has a lemma not equal to its surface form?
- Find the number of words in each POS and sort from most to least frequent (for both English and Chinese)

You can have a list in your condition

- Tell me more about common nouns

```
SELECT count(word), count(DISTINCT word),
       MIN(LENGTH(word)),
       MAX(LENGTH(word)), AVG(LENGTH(word))
FROM word
WHERE pos IN ('NN', 'NNS')
```

count	distinct	min	max	avg
14457	3593	1	21	6.74

- `MIN(num)`, `AVG(num)`, `MAX(num)` are calculated over the whole set

- `DISTINCT` eliminates duplicate records thus fetches only unique records

This list can be the result of a select!

- Tell me more about frequent common nouns

```
SELECT count(word), count(DISTINCT word),
       MIN(LENGTH(word)),
       MAX(LENGTH(word)), AVG(LENGTH(word))
FROM word
WHERE word
IN (SELECT word
    FROM word
    WHERE POS in ('NN', 'NNS')
    GROUP BY word
    ORDER BY COUNT(word) DESC
    LIMIT 10)
```

count	distinct	min	max	avg
1096	10	3	10	5.13

- First find the ten most common words, then do things to them
  - the trick is to write simple queries first, and then combine them
- Note that more common words are shorter (as we would expect)

Now try to do some corpus queries yourself!

You can Create Tables

You tell the database what the data will look like:

```
CREATE TABLE word (
    sid INTEGER,
    wid INTEGER,
    word TEXT,
    pos TEXT,
    lemma TEXT,
    cfrom INTEGER,
    cto INTEGER,
    comment TEXT,
    usrname TEXT,
    PRIMARY KEY (sid, wid),
    FOREIGN KEY(sid) REFERENCES sent(sid)
    );
```

You can Drop Tables

You tell the database what the data will look like:

```
DROP TABLE word;
```

But be careful, you can't get it back.

You can Insert Data

```
INSERT INTO word (sid, wid, word, lemma, pos)
VALUES (1, 1, 'The', 'the', 'DT');
```

Normally you would add data using a program, or read it in from some other file, ...

You can Update Data

```
UPDATE word SET lemma='a', word='A'
WHERE sid=1,wid=1;
```

This changes the value permanently for all rows that match the condition.

```
UPDATE word SET pos='ART'
WHERE pos = 'DT';
```

You can change a lot at a time.

You can Delete Data

```
DELETE FROM word WHERE word='gannet';
```

But be careful, you can't get it back.

You can Index Data

Indexes are special tables that the database can use to speed up data retrieval. An index is a pointer to data in a table, think of it as index in the back of a book. An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

```
CREATE INDEX word_word_idx ON word (word);
CREATE INDEX word_lemma_idx ON word (lemma);
```

This makes it possible to look up words and lemmas very fast, but makes the database bigger. You normally add them after you have added the data. Whether they speed things up or not is an empirical question, and should be tested.

SQL and NLP

- RDMS and SQL are the backbone of most data storage
- There are many implementations:
  - SQLite, PostgreSQL, ORACLE, MySQL, MS SQL Server, ...
  - typically share the same core
  - may have different extensions
- Text to SQL query is a popular task
  - *Which is the longest verb?*

```
→     SELECT word, pos FROM word
      WHERE POS GLOB 'V*' AND
      LENGTH(word) =
      (SELECT MAX(LENGTH(word)) FROM word
      WHERE POS GLOB 'V*');
```