Tutorial 6: Corpus Querying

Informatics 1 Data & Analysis — Tutorial Notes

Week 8, Semester 2, 2016/17

This worksheet has three parts: tutorial *Questions*, followed by some *Examples* and their *Solutions*.

- Before your tutorial, work through and attempt all of the Questions in the first section. If you get stuck or need help then ask a question on *Piazza*.
- The Examples are there for additional preparation, practice, and revision.
- Use the Solutions to check your answers, and read about possible alternatives.

You must bring your answers to the main questions along to your tutorial. You will need to be able to show these to your tutor, and may be exchanging them with other students, so it is best to have them printed out on paper.

If you cannot do some questions, write down what it is that you find challenging and use this to ask your tutor in the meeting.

Tutorials will not usually cover the Examples, but if you have any questions about those then write them down and ask your tutor, or post a question on *Piazza*.

It's important both for your learning and other students in the group that you come to tutorials properly prepared. Students who have not attempted the main tutorial questions will be sent away from the tutorial to do them elsewhere and return later.

Some exercise sheets contain material marked with a star $\star$. These are optional extensions.

Data & Analysis tutorials are not formally assessed, but the content is examinable and they are an important part of the course. If you do not do the exercises then you are unlikely to pass the exam.

Attendance at tutorials is obligatory: if you are ill or otherwise unable to attend one week then email your tutor, and if possible attend another tutorial group in the same week.

Please send any corrections and suggestions to Ian.Stark@ed.ac.uk

Introduction

In this tutorial you will be working with a real corpus and analysing it with the *Corpus Query Processor* (CQP), a query engine that searches corpora based on user queries about words, parts of speech, or other markup.

Reading

The following exercises are based largely upon a tutorial written by Stefan Evert and the other maintainers of the *IMS Open Corpus Workbench* at http://cwb.sourceforge.net. Before starting these exercises, you must read Sections 1 and 2 of the *CQP Query Language tutorial*. Download this as a PDF document from the course web page, which hosts the correct version for this worksheet.

Setting up the working environment

These exercises must be carried out on one of the DICE machines, as those have access to both the CQP engine and the necessary corpora. In particular, you will be using a corpus comprising the works of Charles Dickens, annotated with part-of-speech tags and syntactic structure.

DICE machines in the computing labs are available 24/7. If connecting remotely, please log in first to student.ssh.inf.ed.ac.uk and then student.compute which has CQP installed. A command-line connection through ssh or PuTTy should be sufficient. See the Informatics Computing Support help pages for more information on remote working.

You must first declare the location of a corpus index. Type the following at the command line in a terminal window:

```
export CORPUS_REGISTRY=/afs/inf.ed.ac.uk/group/teaching/inf1da/corpora/registry
```

Take care to copy this exactly.

To start CQP itself, enter the command '`cqp -e`'. (The annotation '`-e`' enables line-editing; try '`cqp -eC`' for colour.) You should then see the following in your command prompt:

```
[no corpus]>
```

This means that `cqp` is active but no specific *corpus* has been selected yet. In order to select the `DICKENS` corpus type

```
[no corpus]> DICKENS;
```

You will then see the following:

```
DICKENS>
```

Command lines in `cqp` finish with a semicolon '`;`'. If you leave it out, though, the tool will usually still accept your input.

You can now get more specific information about this particular corpus by typing:

```
DICKENS> info;
```

Press the space bar to display the next page and '`q`' to get back to the command prompt. To leave `cqp`, enter the command '`exit;`'.

1 Concordances

1.1 Concordance of a given word

Having set up a working environment, we can start performing some queries on the DICKENS corpus. Searching for *concordances* (all occurrences of a given word, displayed in context) is a

basic form of querying a corpus. To search for a specific word, such as "dear", you should type:

```
DICKENS> [word = "dear"];
```

which can be abbreviated to

```
DICKENS> "dear";
```

As previously, press space for the next page and '`q`' to get back to the command prompt.

We can make any search case insensitive with the '`%c`' modifier:

```
DICKENS> "dear" %c;
```

We can search for more than one word after another by adding more queries:

```
DICKENS> "dear" "friend";
```

1.2 More complex queries

The use of *regular expressions* considerably extends the power of the CQP query language. CQP uses the following format for regular expressions:

`exp1 exp2`: first `exp1` then `exp2` in sequence
`exp*`: zero or more occurrences of `exp`
`exp?`: zero or one occurrences of `exp`
`exp+`: one or more occurrences of `exp`
`exp1|exp2`: either `exp1` or `exp2`

For example, the following query retrieves four variations of the word "regular": regular, regulars, regularly, regularity.

```
DICKENS> "regular(|s|ly|ity)";
```

A single dot ('`.`') can be used as a wildcard in CQP, matching any single character. For example, this query finds words such as "than", "then" and "thin":

```
DICKENS> "th.n";
```

while the following query finds words starting "un..." and ending "...ly", , such as "unusually" and "unsuccessfully":

```
DICKENS> "un.*ly";
```

The use of boolean logic extends further the power of CQP queries. The operators '`&`' (and) and '`|`' (or) can be used to combine two tests, while '`!=`' tests for inequality. For instance, the following query would find all occurrences of words beginning "un" and ending "ly", but different from "unlikely":

```
DICKENS> [(word = "un.*ly") & (word != "unlikely")];
```

Similarly, the following query would find all occurrences of the words "man" and "woman":

```
DICKENS> [(word = "man") | (word = "woman")];
```

Question 1: Concordance of a given bigram. A *bigram* is any two-word sequence in some text. Retrieve all occurrences of the bigram "great deal" in the Dickens corpus. Is this query affected by making it case insensitive?

(Tutor Notes) We can use

```
DICKENS> "great" "deal";
```

and for a case insensitive query:

```
DICKENS> "great" %c "deal" %c;
```

Yes, by making this query case insensitive more variations of the bigram "great deal" are retrieved.

Question 2: Concordance of variations of a given word. Retrieve variations of the verb "sit" in the corpus: "sits", "sat", "sitting", ... Is this query affected by making it case insensitive?

(Tutor Notes) Either the simple

```
DICKENS> "(sit|sat|sits|sitting)";
```

or something more complex like

```
DICKENS> "s(it(|s|ting)|at)";
```

Yes, by making this query case insensitive more occurrences like "Sitting" are retrieved.

```
DICKENS> "(sit|sat|sits|sitting)" %c;
```

2 Annotations

2.1 Displaying linguistic annotations

When working with `cqp` you may want to customise how the results of queries are displayed. For example, we can select whether or not to display part-of-speech (POS) annotations on the corpus. By default these are turned off, but they can be enabled with

`DICKENS> show +pos;` (Query results will *show* annotations)

and disabled with

`DICKENS> show -pos;` (Query results will *hide* annotations)

Try some of your queries again, and look at the annotations.

2.2 Accessing linguistic annotations

Once we have annotations on text, we can use these make more refined searches. For example, with the POS annotations we can search for uses of adjectives, verbs or nouns in a given corpus. Switch on POS tagging with "`show +pos;`" and do some searches to identify the different annotations that used for verbs, adjectives and nouns.

These tags are taken from the *Penn Treebank* project. You can find a list of tags online at http://is.gd/treebanktagset together with annotation guidelines describing how they are to be used. For more about treebanks, see the Wikipedia article https://en.wikipedia.org/wiki/Treebank.

In the same way that a query `[word = "Dear"]` will find a particular word in the corpus, a query `[pos = "NN"]` will find all (singular) nouns in the corpus. Both "word" and "pos" queries can use regular expressions and logical operations to build more complex queries.

Question 3. What is the common pattern for the POS annotations used for different types of verbs? Use a regular expression to formulate a query that retrieves all occurrences of verbs in the corpus.

(Tutor Notes) All verb annotations found in the corpus (i.e. VB, VBD, VBG, VBN, VBP, VBZ) start with VB, followed by zero or more characters.

```
DICKENS> [pos = "VB.*"];
```

Question 4. Write a query to retrieve all tokens which are *not nouns*.

(Tutor Notes) `DICKENS> [pos != "N.*"];`

⋆ **Question 5.** Retrieve all occurrences of the word "lock" where it is used as a noun.

(Tutor Notes) `DICKENS> [(pos = "N.*") & (word = "lock" %c)];`

Case-insensitivity turns out to make a real difference here — Plashwater Weir Mill Lock on the Thames plays a role in Dickens's *Our Mutual Friend*.

⋆ **Question 6.** The "`lemma`" annotation in `cqp` contains the *headword*: the principal dictionary word associated with a word. Switch on lemma tagging with "`show +lemma;`" and try some queries.

Use this to formulate an alternative (case-insensitive) query for all variations of the verb "sit" in the corpus. Does this return the same results as your answer to Question 2?

(Tutor Notes) `DICKENS> [lemma = "sit" %c];`

This does return exactly the same words as in Question 2. One way to check is to observe that the following query returns no matches at all

```
DICKENS> [(lemma = "sit" %c) & (word !=  "(sit|sat|sitting|sits)" %c) ];
```

3 Frequencies of a given word

Frequency information obtained from corpora can be useful for language research. Here we look at finding the absolute frequency — the number of occurrences — of certain features in the corpus.

To find the total number of tokens in the corpus, type this:

```
DICKENS> Q1 = []; size Q1;
```

Here we have named the result of the query `Q1`, and then retrieved its size. In this case, the query is one that matches every word in the corpus (and, in fact, every punctuation mark too).

A similar query counts the absolute frequency of a specific word:

```
DICKENS> Q2 = [word = "dear"]; size Q2;
```

This query finds the absolute frequency of all bigrams consisting of the word "dear" followed by a noun:

```
DICKENS> Q3 = [word = "dear"] [pos = "N.*"]; size Q3;
```

We can refine this to list the absolute frequency of the bigram grouped by which noun it is that follows "dear":

```
DICKENS> Q3 = [word = "dear"] [pos = "N.*"]; count Q3 by word;
```

You can also group by POS, or lemma; or use `%c` to ignore case.

A tricky point: Suppose we want to find the most common word in the corpus. A natural query would be:

```
DICKENS> Q4 = []; count Q4 by word;
```

This works, but also includes punctuation marks and white space amongst words. Instead, the following works better:

```
DICKENS> Q4 = [word = "[a-zA-Z].*"]; count Q4 by word;
```

Here the range expression `[a-zA-Z]` matches any letter, either lower or upper case.

Question 7 - Absolute Frequency. Find the absolute frequency of the word "girl" in the corpus, ignoring any case differences.

(Tutor Notes) `DICKENS> Q = [word = "girl" %c]; size Q;`

The frequency of the word "girl" in the Dickens corpus is 1007.

Question 8 - Relative Frequency. Calculate the relative frequency of the word "girl" in the corpus (the number of occurrences as a fraction of the total number of words). Note that `cqp` doesn't include arithmetic, so you will need to do this outside the program.

(Tutor Notes) The relative frequency of the word "girl" is $f(girl)/N = 1007/2816860 = 0.0003575 = 0.036\%$.

In past years some students attempt to do the arithmetic entirely inside CQP; I don't think this is possible — you need to obtain the two counts and then divide one by the other using something else.

⋆ **Question 9.** Find the most common bigram in the Dickens corpus.

(Tutor Notes) This query

```
DICKENS> Q = [word = "[a-zA-Z].*"] [word = "[a-zA-Z].*"];
DICKENS> count Q by word;
```

reports that the most common bigram in the Dickens corpus is "of the", with absolute frequency 14748.

Note that the query "`Q = [][]; count Q by word;`" finds all pairs of adjacent tokens, including punctuation symbols, and would therefore be wrong for this question.

4 Collocations

A *collocation* is a sequence of words that occur in text more often than would be expected by chance; more generally, that form expressions through repeated use in certain contexts.

Question 10. List the 10 most common adjective-noun pairs in the Dickens corpus. Which of these would you classify as collocations?

(Tutor Notes)

```
DICKENS> Q = [pos = "JJ.*"] [pos = "N.*"];
DICKENS> count Q by word;
```

The 10 first results of the above query are:

```
781     old man  [#68388-#69168]
621     young lady  [#112350-#112970]
575     old gentleman  [#67050-#67624]
473     young man  [#113014-#113486]
422     young gentleman  [#111512-#111933]
396     old lady  [#67925-#68320]
309     old woman  [#70055-#70363]
283     long time  [#54675-#54957]
253     last night  [#46894-#47146]
241     great deal  [#35038-#35278]
```

Among those bigrams two appear to be candidate collocations: "great deal" (when used to mean "a lot") and possibly "old man" (when used to mean "father"). Other possibilities of collocations include "last night" and "long time".

Question 11. For any specific adjective-noun pair AB (i.e., adjective A followed by noun B) we can create a *contingency table* of frequency observations. Such contingency tables will be used later in the course. The table for AB looks like this:

	A	$\neg A$	Total
B	$F_{A\,B}$	$F_{\neg A\,B}$	F_B
$\neg B$	$F_{A\,\neg B}$	$F_{\neg A\,\neg B}$	$F_{\neg B}$
Total	F_A	$F_{\neg A}$	F

where:

$F_{A\,B}$ is the absolute frequency of adjective A followed by noun B
$F_{\neg A\,B}$ is the absolute frequency of any adjective *except* A followed by noun B
F_B is the absolute frequency of the noun B
$F_{A\,\neg B}$ is the absolute frequency of adjective A followed by any noun *except* B
$F_{\neg A\,\neg B}$ is the absolute frequency of any not-A adjective followed by any not-B noun
$F_{\neg B}$ is the absolute frequency of nouns other than B
F_A is the absolute frequency of adjective A
F_A is the absolute frequency of adjectives other than A
F is the absolute frequency of adjective-noun pairs in the corpus.

Construct the contingency table of case-insensitive frequency observations for the bigram "great deal". Does this table tell us anything notable about the bigram?

(Tutor Notes) We start with

```
DICKENS> QAB = [pos = "JJ.*" & word = "great" %c] [pos = "N.*" & word = "deal" %c];
DICKENS> size QAB;
```

This finds that $F_{A\,B} = 242$.

```
DICKENS> QNAB = [pos = "JJ.*" & word != "great" %c] [pos = "N.*" & word="deal" %c];
DICKENS> size QNAB;
```

This gives $F_{\neg A\,B} = 215$.

```
DICKENS> QANB = [pos = "JJ.*" & word = "great" %c] [pos = "N.*" & word != "deal" %c];
DICKENS> size QANB;
```

From this $F_{A\,\neg B} = 2776$.

```
DICKENS> QNANB = [pos="JJ.*" & word!="great"%c] [pos="N.*" & word!="deal"%c];
DICKENS> size QNANB;
```

This reports $F_{\neg A\,\neg B} = 111047$.

The final contingency table is as follows — the row and column totals can be calculated either by addition or with further `cqp` queries.

	great	¬great	Total
deal	242	215	457
¬deal	2776	111047	113823
Total	3018	111262	114280

The point of the contingency table is to gather statistical information for detecting whether the bigram "great deal" occurs in the corpus with atypically common frequency. This information can be used to determine whether "great deal" is a collocation. We can see here, for example, that more than half of the time "deal" is paired with an adjective, that adjective is "great". This does seem quite high at first sight.

To explore this properly, though, requires statistical techniques that can estimate the chance of such a distinctive feature arising purely by chance. This is the topic of χ^2 (chi-squared) testing that is covered later in the course.

5 Tutorial Discussion

In this tutorial we have used `cqp` to explore the characteristics of a small corpus taken from the works of Charles Dickens. This contains something over 3 million words. That's a lot for a human to read; and even for a computer it's large enough to observe that some queries run faster than others. It's also large enough that a poor choice of algorithms could make even a simple query take years to complete.

How do you think `cqp` works? What algorithms and techniques might it use to answer the queries you have put to it during this tutorial?

(Tutor Notes) There are lots of directions this discussion might take. Some of the material from other courses might be relevant, such as finite automata from *Inf1-CL* for efficiently matching regular expressions in a single pass through text. That turns up at two levels here: in matching words against regular expressions `(sit|sat|sits|sitting)` or in searching for grammatical patterns like `[pos="JJ.*"]+ [pos="N.*"]` for multiple adjectives followed by a noun.

The other compulsory Informatics courses haven't yet done much about complexity classes, so it's hard to discuss directly here. However, there are still some clear examples of slow and faster ways to do things. If we needed to collect a count of all the different words in the corpus, then one can imagine two different approaches.

- For each word, go through the corpus counting how many times it occurs.
- Make one pass through the corpus to build up a record for all words at once: for each word, increment its counter in the record.

Depending on the size of the corpus and the number of different words, it's possible that the second method will be feasible while the first is impossibly slow.

Even when the entire corpus can be loaded into a computer's memory — all of the material for the `cqp` Dickens corpus is 64MB — some computations will still be infeasibly slow: for example, making a separate pass over the data for every one of the 3M words in the corpus.

- Given text files containing three million words, how could you efficiently search for all occurrences of the word "tomorrow"?

(Tutor Notes) Here are a couple of small steps that might help to search for all occurrences of the word "tomorrow".

– Simplest is to observe that if we see a first letter that's not "t", it isn't worth going on to check other letters in the word: it's definitely not the one we want.

– Less obviously, if we have a stream of characters not yet broken into words, then it can be faster to jump ahead and look for the *end* of the word. Instead of looking for "t" at position n, look at the letter in position $(n + 7)$. If it turns out to be "k", then we aren't looking at any part of "tomorrow", so we can jump again to $(n+14)$ and be sure we haven't missed anything.

This idea is the basis of the *Boyer-Moore* string-matching algorithm.

- Could this search be made faster using several computers in parallel?

(Tutor Notes) Yes, this search will parallelize very well: with n processors, break the corpus up into n blocks of equal size and search them all in parallel. Some care is needed at the ends, to make sure nothing is missed across the points where the corpus is split.

Almost everything that we've done with `cqp` here will parallelize in a straight forward way to give a direct speedup.

- Suppose you are writing a program that will be used to repeatedly search for different words in these files. Is there any pre-processing you could carry out that might be expensive to do once but would help make later searches faster? What information would you collect?

(Tutor Notes) Building an index is almost always a helpful step in processing text corpora: it may be expensive to do once, but speed up many later searches.

A traditional index in a book usually only includes important topic words and the pages where they appear. A search index for a corpus is much more comprehensive: for every single word, it lists the exact location of every occurrence of that word. This kind of *inverted index* may end up larger than the document itself: after all, it contains enough information to completely reconstruct the original document.

Some corpora will have several such indexes: every word, every kind of part of speech, and so on for other annotations. These may also use complex data structures to speed up access to elements within the access.

Once these indexes are prepared, some simple searches might now be done in a single retrieval from the index, without any need to inspect the corpus itself.

- What about more complex queries, such as counting the absolute frequency of all bigrams in the corpus — can you think of a way to implement those efficiently?

(Tutor Notes) I can't think of a way an index would help much with building a table to record occurrences of all bigrams in a corpus. However, it is still something that can be constructed in a single pass through the corpus: accumulating a record mapping bigrams to counts. Some care might be required in the choice of datastructure for that record — it's important to be able to identify quickly whether or not a bigram has already appeared, and either add it to the record or update its count so far.

Examples

This section contains further exercises on Corpus Querying. These examples are similar to the main tutorial questions: they involve querying the Dickens corpus with CQP, and they cover topics like concordances, annotations, word frequencies and collocations. Instructions on how to use CQP are provided in the tutorial introduction.

Example 1: Concordances

(a) Retrieve all occurrences of the word "proud" in the Dickens corpus.

(b) Retrieve all occurrences of the word "proud" where the query is case-insensitive.

(c) Retrieve variations of the word "proud" in the corpus: "proudly", "proudest", ...

(d) Retrieve all occurrences of the bigram "sweet girl" in the corpus.

Example 2: Annotations

(a) Write a query to retrieve all adjective occurrences in the corpus.

(b) Write a query to retrieve all tokens which are adjectives or adverbs.

(c) Retrieve all occurrences of the word "face" where it is not used as a noun.

Example 3: Word frequencies

(a) Find the absolute frequency of the word "face" in the Dickens corpus.

(b) Find the absolute frequency of the word "face" in the Dickens corpus, where it is not used as a noun.

(c) Find the absolute frequency of all bigrams consisting of an adjective followed by the word "girl".

(d) Find the absolute frequency of all bigrams consisting of an adjective followed by the word "girl", grouped by the adjective that precedes "girl".

Example 4: Collocations

Below you can see a list of bigrams in the Dickens corpus, ordered by their frequency.

```
111     next morning  [#64236-#64346]
100     little while  [#53448-#53547]
97      other people  [#73043-#73139]
96      other side  [#73396-#73491]
94      own room  [#75644-#75737]
93      many times  [#57550-#57642]
88      old friend  [#66892-#66979]
```

```
85      such things  [#98498-#98582]
85      young friend  [#111398-#111482]
78      young men  [#113500-#113577]
76      long way  [#55010-#55085]
```

For each of the following bigrams, use CQP to construct a concordance for the word pair, and assess whether you think it forms a collocation.

(a) "old friend"

(b) "other people"

(c) "next morning"

Solutions to Examples

These are not entirely "model" answers; instead, they indicate a possible solution. Remember that not all of these questions will have a single "right" answer. If you have difficulties with a particular example, or have trouble following through the solution, please raise this as a question in your tutorial.

Solution 1

(a) Retrieve all occurrences of the word "proud" in the Dickens corpus.

```
DICKENS> [word = "proud"];
```

which can be abbreviated to

```
DICKENS> "proud";
```

(b) Retrieve all occurrences of the word "proud" where the query is case-insensitive.

```
DICKENS> "proud" %c;
```

(c) Retrieve variations of the word "proud" in the corpus: "proudly", "proudest", ...

```
DICKENS> "proud(|er|est|ly)";
```

or

```
DICKENS> "proud|prouder|proudest|proudly";
```

or

```
DICKENS> "proud.*";
```

(d) Retrieve all occurrences of the bigram "sweet girl" in the corpus.

```
DICKENS> "sweet" "girl";
```

or

```
DICKENS> [word = "sweet"] [word = "girl"];
```

Solution 2

(a) Write a query to retrieve all adjective occurrences in the corpus.

```
DICKENS> [pos = "JJ.*"];
```

(b) Write a query to retrieve all tokens which are adjectives or adverbs.

```
DICKENS> [(pos = "JJ.*") | (pos = "RB.*")];
```

(c) Retrieve all occurrences of the word "face" where it is not used as a noun.

```
DICKENS> [(word = "face") & (pos != "N.*")];
```

Solution 3

(a) Find the absolute frequency of the word "face" in the Dickens corpus.

```
DICKENS> Q1 = [word = "face"]; size Q1;
```

(b) Find the absolute frequency of the word "face" in the Dickens corpus, where it is not used as a noun.

```
DICKENS> Q2 = [(word = "face") & (pos != "N.*")]; size Q2;
```

(c) Find the absolute frequency of all bigrams consisting of an adjective followed by the word "girl".

```
DICKENS> Q3 = [pos = "JJ.*"] [word = "girl"]; size Q3;
```

(d) Find the absolute frequency of all bigrams consisting of an adjective followed by the word "girl", grouped by the adjective that precedes "girl".

```
DICKENS> Q4 = [pos = "JJ.*"] [word = "girl"];
DICKENS> count Q4 by word;
```

Solution 4

We cannot conclude with certainty whether any of the bigrams provided are statistically true collocations — whether they actually occur more often than would be expected by chance, given the underlying frequency of individual words in the corpus — unless we do some further analysis, such as χ^2 testing (pronounced "kye-squared" — we will discuss these later on in this course). However, by looking at examples of uses of the bigrams in the corpus we can argue whether these bigrams have a semantic weight of their own, and hence whether they could be considered to be collocations.

(a) "old friend"

This bigram has in some cases the meaning of "close friend" or "beloved friend". A representative example in the corpus is "Well met, my dear old friend! said I ." In these cases, "old friend" could be regarded as a collocation.

(b) "other people"

This is not a collocation, as the expression simply refers to other people.

(c) "next morning"

This bigram is used in some cases in the corpus to mean "tomorrow". In these cases it might be considered to be a collocation.