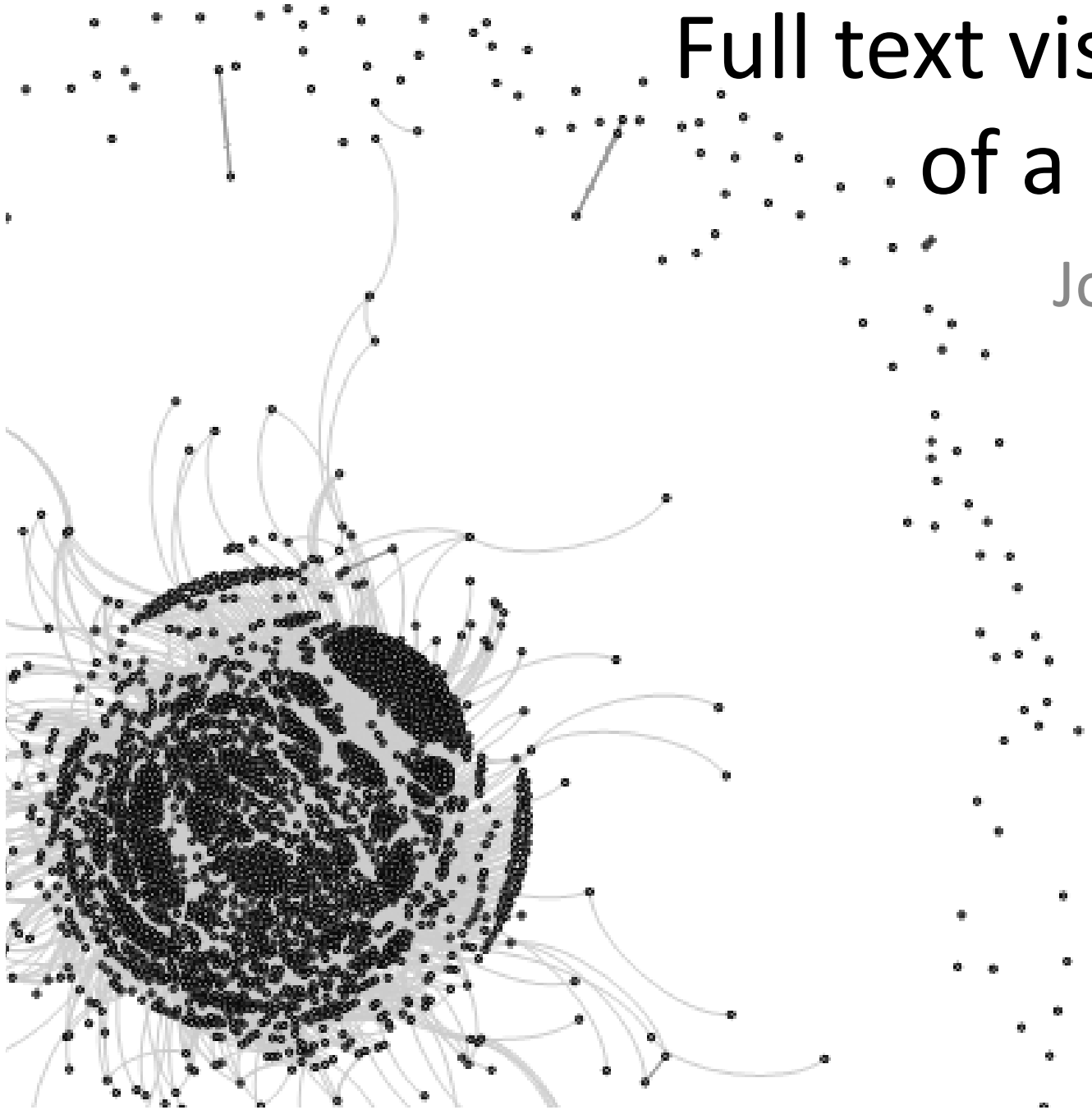

Full text visualisation of a pdf library

Jonathan Street

The problem

- Gathered a large library of pdf files over several years
- Regularly visit about a dozen
- Rubbish memory -> no idea what is in the rest

Options

- Search
  - Works when I know what I’m looking for
  - Dependent on consistent terminology
- Full text visualisation
  - Discovery focused
  - Reduced reliance on consistent terminology

Challenges

- Text extraction
- Calculation of similarity
- Visualisation

Text extraction

- Pdf docs are a pain
- Currently using pdfminer
- Extracted text is usually in a weird order
- Would like to detect
  - Title
  - Section headings
  - Abstract

```
 8
 9 def process_pdfs(rootpath):
10     """Walk through all directories beneath rootpath and extract text from any pdf files found"""
11     timeouts = []
12     errors = []
13     for root, dirs, files in os.walk(rootpath):
14         directory = Directory(root)
15         db_session.add(directory)
16         db_session.commit()
17         print 'Converting files in: %s' % (root)
18         for name in files:
19             if name[-4:] == '.pdf':
20                 try:
21                     if already_processed(name, root):
22                         print 'Already converted: %s' % (name)
23                         continue
24                     print 'Converting: %s' % (name)
25                     doc = Document(name)
26                     doc.directory_id = directory.id
27                     q = multiprocessing.Queue()
28                     p = multiprocessing.Process(target=extract_pages, args=(os.path.join(root, name), q,))
29                     p.start()
30                     p.join(timeout=5*60)
31                     if p.is_alive():
32                         # Timeout has expired
33                         p.terminate()
```

```
63
64 def extract_pages(filename, q):
65     """Extract text from the pdf file filename"""
66     try:
67         pages = pdf.get_pages(filename)
68         q.put(pages)
69     except:
70         q.put(False)
71
```

```
147
148 def parse_lt_objs (lt_objs, page_number, images_folder, text=[]):
149     """Iterate through the list of LT* objects and capture the text or image data contained in each"""
150     text_content = []
151
152     page_text = {} # k=(x0, x1) of the bbox, v=list of text strings within that bbox width (physical colu
153     for lt_obj in lt_objs:
154         if isinstance(lt_obj, LTTextBox) or isinstance(lt_obj, LTTextLine):
155             # text, so arrange is logically based on its column width
156             page_text = update_page_text_hash(page_text, lt_obj)
157         elif isinstance(lt_obj, LTImage):
158
159             pass
160
161         elif isinstance(lt_obj, LTFigure):
162             # LTFigure objects are containers for other LT* objects, so recurse through the children
163             text_content.append(parse_lt_objs(lt_obj, page_number, images_folder, text_content))
164
165     for k, v in sorted([(key,value) for (key,value) in page_text.items()]):
166         # sort the page_text hash by the keys (x0,x1 values of the bbox),
167         # which produces a top-down, left-to-right sequence of related columns
168         text_content.append(''.join(v))
169
170     return '\n'.join(text_content)
171
172
```

Similarity

- Cosine similarity on Term Frequency – Inverse Document Frequency (TF-IDF) vectors
- Two stage process
  - Dictionary creation
  - Measure TF and generate TF-IDF

TF-IDF

- TF
  - How often does the term appear in this document
- IDF
  - What fraction of the docs in the library contain this term
  - Common words have a very low value, rare words have a higher value

Cosine similarity

Term	the	share	cliff	bluefish
Document 1	0.001	0.16	14.4	0.0
Document 2	0.0013	0.32	1.5	32.6
Cosine similarity	1.3e-6	0.0512	21.6	0.0

```
59 def build_IDF_dict(stopwords, reader=UnicodeReader(), stemmer=Stemmer()):
60     """Optimised for low memopry usage"""
61     docs = Document.query.all()
62     print 'query'
63     corpus_size = len(docs)
64     scale = math.log(corpus_size)
65     term_count = collections.defaultdict(int)
66     dictionary = {}
67     print 'setup size and vars'
68     for j, doc in enumerate(docs):
69         print '%d papers processed' % j
70         text = ' '.join([i.body for i in doc.pages.all()])
71         words = [w.stem for w in map(stemmer, reader(text))]
72         words = set(words)
73         for w in words:
74             term_count[w] += 1
75
76     print 'terms have been counted'
77     for w, cnt in term_count.iteritems():
78         if cnt > 2:
79             dictionary[w] = math.log(corpus_size / (cnt + 1)) / scale
80
81     return dictionary
82
```

```python
from database import db_session
from models import *

import cPickle as pickle
from tagger import tagger

docs = Document.query.all()

weights = pickle.load(open('tagger/data/dict.pkl', 'rb')) # or your own dictionary
myreader = tagger.Reader() # or your own reader class
mystemmer = tagger.Stemmer() # or your own stemmer class
myrater = tagger.Rater(weights) # or your own... (you got the idea)
mytagger = Tagger(myreader, mystemmer, myrater)


for doc in docs:
    txt = '\n\n'.join([i.body for i in doc.pages])
    tags = mytagger(txt, 5)
    for tag in tags:
        qry = Tag.query.filter_by(Tag.tag == tag).first()
        if qry:
            doc.tags.append(qry)
        else:
            doc.tags.append(Tag(tag))
    db_session.add(doc)
    db_session.commit()
```

Limitations

- Bag of words
  - I have six apples, you have two
  - I have two apples, you have six

Visualisation

- Ideally have a graph embedded in a stand alone program or a web page
- Currently using gephi

Refs

- Text extraction from pdf docs
  - http://denis.papathanasiou.org/?p=343
- Inspiration
  - http://jonathanstray.com/a-full-text-visualization-of-the-iraq-war-logs
- Tagging
  - https://github.com/apresta/tagger
  - https://github.com/Torkn/tagger
- Gephi
  - http://gephi.org/