Applied Topic Models

Daniel Lawson — University of Bristol

Lecture 07.2 (v1.0.1)

Signposting

- This is a continuation of Topic Models, now with a focus on how we make them work in practice.
  - This is not trivial and includes a lot of tradecraft.
  - Not all of this is language agnostic.
  - Performance and generalisability can be improved dramatically by tailoring to the target data.

ILOs

- ILO1 Be able to access and process cyber security data into a format suitable for mathematical reasoning
- ILO2 Be able to use and apply basic machine learning tools

Data Quality

Garbage in - GarbaHTFGNK KGDFgfdggggggg

Cleaning (Text) data

- This course is about cyber data.
- Topic modelling can be applied to many cyber datasets without there being actual text.
- However, some cyber data contains text, and some cyber problems involve text.
  - For example, detecting phishing.
- So we'll cover the basics of text cleaning.
- You need to know the basics of **regular expressions** to cut the text down to the core text.
- Regular expressions are a very general syntax for specifying search patterns.

Data cleaning pipeline

- Remove the **punctuation** marks: ',.;:?!'
- Remove the **stop-words**, like "I", "and", and "the"
- Remove too **common words**
- **Standardize** spacing: double spaces, tabs, newlines
- What do you want to do with special words and characters? e.g. Twitter "rt", "@user", "#hashtag!"
- Correct cleaning is **context specific**.
  - Legal documents are different to tweets, html, blog posts, etc!
- It is unlikely that the same subject discussed in two different fora will look the same to a topic model!

Data from unusual sources

- Use a converter to 'plain text':
- textract:

```
### **textract** for converting from a wide
### range of sources including MS and pdf
import textract
text = textract.process("path/to/file.extension")
```

Data from unusual sources

- Use a converter to ‘plain text’:
- textract:

```python
### **textract** for converting from a wide
### range of sources including MS and pdf
import textract
text = textract.process("path/to/file.extension")
```

- pdfminer:

```python
### dedicated tool: should be better performance
import pdfminer
convert_pdf_to_txt('file name')
```

Cleaning (Text) data

- Identify or remove special words (emoticons, hashtags),
- Remove common words (“stop words”),
- Lemmatise or stem (standardize endings),
- Where multiple meanings exist, use context to deduce correct one (noun/verb/adjective?).
- We cover these details in the workshop.

Regexp

- Essential for pre-cleaning your data.
- See the Python Documentation.
- Regular expressions can contain both special and ordinary characters.
- Most ordinary characters, like ‘A’, ‘a’, or ‘0’, are the simplest regular expressions; they simply match themselves.
- Some characters, like ‘|’ or ‘(’, are special.
- Special characters either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted.
- Repetition qualifiers (*, +, ?, {m,n}, etc) define how many characters are wanted.

Regexp in python

- Basic usage:

```
match = re.search(pattern, string)
if match:
    process(match)
```

- Many more complex possibilities exist!
- Search/Replace/Group/Split etc.
- Basic usage is massively helpful.
- Lookup more complex problems.

Regexp special characters

- `\`: Escape special character.
- `.` (dot): match any character
  - `r"me."`: matches the string `men` or `met` but not `me` at the end of a word.
- `^` (caret): start of string
  - `r"^me"`: matches `me` at the start only (`meaning`)
- `$` (dollar): end of string/final character before newline
  - `r"me$"`: matches `me` at the end only (`biome`)
- `*` (star): 0 or more matches of preceding RE
  - `r"file.*\.txt"`: matches all strings of the form "file", anything, and ".txt"
- `+` (plus): 1 or more matches of preceding RE
  - `r"file.+\.txt"`: matches "file", any one character, and ".txt"
- `[]`: Set of characters.
  - `r"file[0-9]+\.txt"`: matches forms like "file5.txt"
- `{m}`: match m copies of receding RE
  - `r"file[0-9]{3}\.txt"`: matches forms like "file005.txt"

Example of cleaning (Text) data with regexp

```
import re
def preprocessor(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',
                           text)
    text = (re.sub('[\W]+', ' ', text.lower()) +
            ' '.join(emoticons).replace('-', ''))
    return text
```

Quantifying solutions

- There are many ways to quantify how good a particular LDA model is. The most popular are:
- **Perplexity**: the perplexity is $2^{-H(D)}$ where $H(D) = \sum_{t=1}^{T} \log(p(t|\theta_d))$
  - $p(t|\theta_d) = \sum_{v=1}^{V} \theta_d(v) p(t|v)$ uses the model-learned topics V for the (held out!) document d with topic distribution θ_d.
  - It is the entropy of term t (normally reported as the average per-word).
  - Perplexity is low (better) when each word appears in only one topic.
  - Perplexity is high when words are distributed across topics.
- **Coherence**: a measure of how often pairs of words appear together. there are two ways to examine this:
  - **intrinsic coherence**: called u_mass, this compares within a corpus.
  - **extrinsic coherence**: called c_v, this compares to some standard reference documents.
- Neither is particularly consistent with human judgement[1].

[1] Chang, Jonathan, Jordan Boyd-Graber, Sean Gerrish, Chong Wang and David M. Blei. 2009. Reading Tea Leaves: How Humans Interpret Topic Models.

Coherence

- The coherence is based on the score [2] (defined next):

$$Coherence(V) = \sum_{(t_i,t_j)\in V} score(t_i, t_j)$$

  - Where V is a topic, and t_i, t_j are word pairs.
- In both cases we use a **regulariser** ϵ.
  - $\epsilon = 1$ is natural but not obligatory.

[2] Stevens, Kegelmeyer, Andrzejewsk and Buttler Exploring Topic Coherence over many models and many topics

intrinsic coherence

- Using the score function:

$$u\_mass(v_i, v_j) = \log\left(\frac{p(v_i, v_j, \epsilon)}{p(v_i)p(v_j)}\right)$$

- i.e. we compare the probability that the words co-occur in a document with their relative frequencies.
- ϵ assigns non-zero weight to word pairs that do not occur together in a document.

extrinsic coherence

- Using the score function:

$$c\_v(v_i, v_j) = \log\left(\frac{D(v_i, v_j, \epsilon)}{D(v_j)}\right)$$

- where D counts documents that contain the word(s);
- i.e. we compare the frequency in which words co-occur in an external dataset, compared to their external frequency.

Reflection

- To what extend can NLP be considered a supervised task?
- What do the scores quantify? How do you externally verify their performance?
- What challenges appear in processing languages that lack word standardization?
- How does this extend to non-language applications of topic modelling?
- By the end of the course, you should:
  - Be able to apply topic models to both cyber security data and text data,
  - Understand its uses and limitations at a high level.

Signposting

- Next lecture: Workshop on NLP.
- Next block: Algorithms Every Data Scientist Should Know:
  - Sampling,
  - Filtering,
  - Sketching,
  - And more!

References (1)

Data science topic modelling

- Preparing Data for Topic Modelling
- NLP for legal documents
- Machine-Learning-In-Law github repo

Judging topic models

- Chang, Jonathan, Jordan Boyd-Graber, Sean Gerrish, Chong Wang and David M. Blei. 2009. Reading Tea Leaves: How Humans Interpret Topic Models. NIPS.
- Stevens, Kegelmeyer, Andrzejewsk and Buttler Exploring Topic Coherence over many models and many topics

References (2)

Data sources

- Kaggle dataset for fake news
- Intelligence and Security Informatics Data Sets
- Vizsec security data collection
- Threatminer cyber data with NLP
- Phishing data corpus with paper A Machine Learning approach towards Phishing Email Detection.